SimEther - A Package for Qualitative Ethernet Simulation

Martin van den Nieuwelaar
Supervisor: Ray Hunt

October 1993

Contents

List of Figures

Chapter 1

Introduction

SimEther is a low level graphical Ethernet simulator.

Its main purpose is to aid the lecturer or teacher in demonstrating the Ethernet protocol, and how the associated network, including cable, and Ethernet devices work. Secondly, it can be used by the student who has some basic familiarity with networking, wishing to learn how Ethernet works, or just as a reference guide.

SimEther is not a quantitative simulator. There is no way to simulate the throughput, or efficiency, or any other quantitative parameter of a network. It is purely for demonstrating the methodology of Ethernet, with a minimum mathematical or statistical involvement.

The report can be broken down into the following chapters:

Chapter 2 – Reasons for the project, and the target environment.
Chapter 3 – The design methodologies used.
Chapter 4 – How the simulation and user interface was constructed
Chapter 6 – What SimEther has achieved
Chapter 7 – Areas in which more work could be applied.

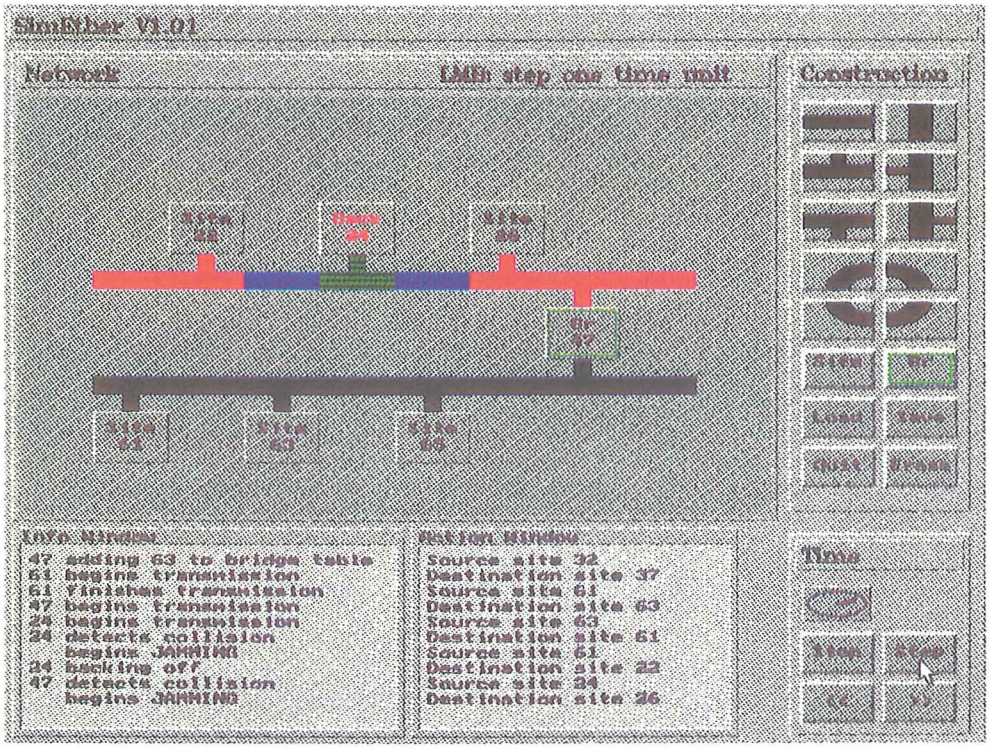

Figure 1.1: Example SimEther screen snapshot

Chapter 2

Reasons for the project

SimEther has been developed to facilitate quicker and more thorough understanding of the Ethernet protocol, than by teaching directly from a textbook.

Computer based educational tools have advantages over the paper equivalent in a number of areas.

- Static diagrams lack what-if scenarios. Should a student ask about a particular situation, the lecturer must try and explain with the aid of a diagram. This is not always entirely successful. With a computer based system – in particular the design methodology used for SimEther – a network can quickly be constructed on the screen. The simulation can be run, and students get a good solid visual representation of what is happening on the network, in addition to an explanation by the lecturer.
- Diagrams tend to have a high level of granularity. In a situation where diagrams are used to explain a system that is varying with time, the points in time where the diagrams are drawn are critical for correct explanation and understanding. Diagrams tend to jump from one important event to another and can lose the flow, or at least scale of events. It may not be clear for example, the amount of time a site is transmitting in proportion to the propagation delay of a frame on a communications network. A computer simulation can demonstrate this explicitly.

An application[1] already in existence for demonstrating the mechanisms of the Token Ring Protocol (IEEE 802.5) has already proven useful. This application has been used typically in addition to paper handouts.

To the best of my knowledge no such equivalent tool for Ethernet was available at the commencement of this project. There already existed tools for performance analysis, and design of Ethernet networks, but nothing for the sole purpose of teaching the operation of Ethernet sites and bridges.

Bearing the advantages of a computer based educational tool in mind, and the fact that no known Ethernet teaching tool currently existed, is was decided that SimEther would be useful. In addition, it would be a companion tool for the Token Ring demonstration program.

[1] IBM Cabling System written by C. Danl Keys in 1984

Chapter 3

Design

3.1 General

When designing an application that is to be used in a teaching role, it is essential that the simulation be correct. Teaching incorrect theory is to be avoided at all cost. At the very least, time will be wasted when the correct theory has to be taught later.

3.2 Portability

The user interface design chosen[1] requires the target platform to be able to display high resolution pictures, and be colour capable.

It was decided to build SimEther on the IBM PC for a number of reasons. PCs are popular in the places where this sort of application would likely be useful[2]. Additionally, the Token Ring demonstration tool which runs on the PC could be used in conjunction with SimEther, on the same machine.

Other platforms were considered but dismissed for various reasons:

[1] See the section on User Interface design (3.4)
[2] Universities and Polytechnics

- Sun Sparc - lack of availability outside of universities, general high cost especially for colour displays, and possibly difficult to obtain datashows.
- Apple Mac - Probably the second most suited. Colour Macs are not very common (though becoming more so). Two machines would be required to demonstrate both Ethernet and Token Ring since the Token Ring package is written for the PC.
- Commodore Amiga - Technically best suited. Every Amiga offers colour output, and can be connected directly to standard video displays (PAL), as opposed to specialised datashows which most other machines require. Very few machines compared to PCs and Macs meant this platform was dismissed.

The language chosen was C++. C is a widely available language, and hence should enable easy porting to other platforms should this be required. Some of the C++ extensions available were used to implement the PC user interface. These extensions would not be used in the general simulation code, to enhance portability.

While the version presented here only runs under MS-DOS[3], the development of SimEther has taken place over a number of platforms including the Amiga (AmigaDOS), and SUN (UNIX). The present version is written in C++, using Borland C++ version 3.0.

3.3 System Requirements

This particular PC implementation of SimEther requires a two button mouse, with associated driver software that must be installed prior to running SimEther.

The program adapts itself to run with most machines offering a bit-mapped screen capability. It should be noted however that while all screen resolutions are supported, SimEther was written using a VGA resolution screen (640 by 480 pixels), and hence it is impossible to show all the detail possible in this mode, in a lower resolution mode. SimEther will work properly in EGA mode

[3] or a shell under Windows

(640 by 350 pixels). Lower resolution modes will appear cluttered as there is a limit to the amount fonts can be scaled, and still remain readable.

SimEther presently requires at least a sixteen colour graphics mode. Presently there is no support for machines with lower colour capability, or monochrome displays. The benefit of colour as a teaching aid outweighs the small extra price for colour VGA hardware. Perhaps the only place where monochrome is still common is on laptop and notebook computers, where colour displays are still expensive.

3.4 User Interface

A user interface was written from scratch. This was done because:

- Library routines could not be found in the time allotted that would provide suitable control.
- Many libraries supported text screens only, and others were not flexible enough to provide the kind of interface SimEther was to offer.
- Should there be something wrong with the underlying interface, there would be little that could be done to correct the fault if third party code was used.
- In the case of a major limitation becoming apparent, it would not be possible to adapt third party code.

Some of the design of the user interface of SimEther has come from observation of the use of the Token Ring tool. This tool, like SimEther is typically shown to a class or group of people, with the aid of a datashow and overhead projector. Such a tool is not used for long periods of time within a class. Typical usage is for short sessions, with long periods between, when other topics are covered. Hence, the interface would have to be intuitive enough for intermittent users (in this case the lecturer) to be able to remember much of the syntax from the last time they used the tool. This is a class of user that has much semantic knowledge, but little syntactic knowledge. They know what they want to show, but the way to persuade the program to do this has not been remembered.

A method known as *direct manipulation* was finally chosen. This style of interface is already used in areas such as air traffic control, video games, and other highly interactive applications. As Ben Shneiderman says in his book *Designing the User Interface* [6], "Direct manipulation is appealing to novices, easy to remember for intermittent users, and with careful design it can be rapid for frequent users." This method of interaction style enables the user to have more freedom of choice. For example, a lecturer wishing to explain a particular point can focus more closely on that idea. A straight demonstration program, with its inflexibility, will display all the steps leading to an event. The user will have to skip through numerous stages to get to the point of interest. A direct manipulation package allows the user to better focus on the task at hand. This is the idea of transparency, as applied to tasks. The interface should effectively disappear, or at least require very little cognitive overhead.

There are a number of problems with direct manipulation however:

- Additional effort in absorbing the rules of the representation.
- Learning the meaning of components in the graphical representation.
- Possible misleading graphic representations.
- Excessive screen display space.
- Experienced users may find it slower than a keyboard interface.

Bearing these disadvantages in mind, and taking into account the advantages (including one quote by Shneiderman "The attraction of systems that use principles of direct manipulation is apparent in the enthusiasm of the users."), it was decided that direct manipulation would be suitable.

Colour was a part of the user interface that was particularly concerning. For some reason, a large number of PC programs make use of the colour palette in an ad-hoc manner, combining large numbers of different colours on the same screen. Frequently the choice of colour seems to bear no resemblance to the program state, and appears as if the programmer has used another colour for the fun of it. The Token Ring demonstration software is a good example of this, as is Box Plot, and many other applications.

Some guidelines regarding colour, set down by Shneiderman in [6] would be followed for the implementation, including:

- Limit the number of colours.
- Recognise the power of colour as a coding technique.
- Be consistent in colour coding.
- Use colour in graphic displays for greater information density.
- Use colour changes to indicate status changes.
- Beware the loss of resolution with colour displays

By combining an aesthetically pleasing window system using shades of grey, coupled with limited use of colours, a productive environment for a user to operate in could be created. Colour could be used only to heighten impact, and not just because it is there.

Chapter 4

Implementation

4.1 Ethernet LAN

SimEther would be implemented in an object-oriented fashion. This was done to reduce the interaction required between different elements of the simulation. The analogy with the real-world Ethernet implementation is close. Sites are entities by themselves. They are connected to an Ethernet cable. By measuring voltages on the cable, they can receive data, and by putting voltages onto the cable, they can transmit. Once they have done this, there is no need for the site to do anything more. The laws of physics take care of the propagation of the actual signal down the wire.

This is the methodology used to design SimEther. Sites place outgoing signals on the wire. Note that two signals are actually places on the wire – one travelling in each direction down the wire[1]. There is a routine that has the task of propagating signals. This routine simply looks at each signal on a wire (each wire can contain an infinite number of discrete signals), and propagates it in the direction it is travelling. Other sites on the cable monitor the data that is passing them and if it destined for that site, then site will record the signals in an incoming buffer. The site is then free to process the data in its own time.

One interesting phenomenon not allowed for is the degradation of signals over

[1]This implies a baseband network topology, where no one site is upstream from another

distance. In the simulation a signal will propagate perfectly. This is adequate for most situations. The only place it is blatantly obvious is when a circular network is created. Signals will loop endlessly without decaying[2]. Since nobody would ever seriously wish to make a circular network, this is no great loss! It will still be visible to the beginner user, why Ethernet will not work in a circular fashion by the way the frames flow round and collide with each other.

To allow straightforward editing and construction of networks, a two dimensional gridded area was decided upon. This would allow realistic networks to be constructed without undue complexity. Each Ethernet device would take up one element of the network grid. From each element, connections to the surrounding four elements would be possible[3]. With the granularity imposed by the grid, smooth animation of packets flowing on the network would be difficult to obtain. Presently, animation is done element by element. Time is effectively *quantised* to the amount of time for a frame to propagate the grid distance. Trying to animate more smoothly would have to involve a kludge of some description, since an intermediate state where a grid element is partially occupied by a frame is not allowed for, and would be difficult to implement successfully. Changing the philosophy of "one Ethernet device, one grid", so that a device took up numerous grid elements would reduce the granularity making the animation smoother, but would add complexity to the design. This approach was not taken because the author felt there would be more success in building a simple system that worked, but with large granularity rather than a complex one that may not have.

Bridges in SimEther are built from sites, with a few extra components thrown in for good measure. To be exact, each bridge is comprised of a bridge table, four sites[4], and a frame buffer for each site. The bridge table lists other sites and to what part of the bridge they are connected. This is updated when other sites on the network transmit – just as in the real-world. Incoming frames received by one site of a bridge are checked against the bridge table. Frames that need to be forwarded to other networks are put in the outgoing buffers for those sites. Other frames are filtered. When a site is waiting, and the cable it is connected to is free, it will transmit any frames it has in its buffer in a first in, first out manner.

[2]Super-conducting cables!

[3]differing by one space in either the x *or* y coordinate

[4]This is a limitation of my grid representation, not bridges in general

The ability for bridges to learn in a fashion identical to that of the real world is particularly pleasing, and serves to demonstrate the operation of Ethernet bridges particularly well.

4.2 User Interface

All routines for the implementation of the user interface were written from scratch, with the exception of the mouse handling routine. A C++ object from [4] was used for this purpose.

The user interface is simple, and very intuitive. A full user interface for an operating system would have to be far more complex, and offer more flexibility. For the purpose of this simulation package the present interface is adequate. Windows for example do not have history buffers, or scroll-bars for example. As much as possible was provided that would be genuinely useful in the time provided.

The SimEther window, Network, Construction, and Time windows are purely decorative items for labelling the items contained within. The Info and action windows are implemented as C++ objects, and are scrolling text windows.

A requester object has been defined for creating temporary overlapping screen objects. The requester object is used to save the contents of an area of the screen for later restoration when the space is finished being used. The *specific info* windows that the user can "pop-up", use the requester object for example, as do the file requester, boolean requester, and acknowledgement requester.

A clock object has been defined for monitoring the passage of time. Time control was a primary design consideration. If the user could not control the passage of time precisely in the simulation, a lot of the benefits would be lost. A facility for stepping time one time unit is provided, as well as a variable rate facility. The rate is controlled in a logarithmic fashion, which offers a suitable range of speeds that can be selected quickly. The clock provides feedback as to the rate of passage of time.

Icons are implemented in a linked list manner. Different types are supported, including toggle switches, momentary buttons, and *network icons*. Network

icons are invisible to the user and are used for tracking mouse positions over a grid, such as the network area.

An information system has been implemented. This displays a message to the user informing them what action will occur should they proceed with a particular mouse action. This is of major benefit to the infrequent user. It is hoped it will provide them with enough information that they can decide whether this is the command they wish to initiate, without having to remember trivial syntactic jargon like "press the left mouse button when over an object to view its contents".

The grid on the main network window helps the user position items correctly the first time. It uses the invisible network icons to locate what element the cursor is above, and highlights that element.

Chapter 5

User Notes

What follows are a few notes that will be of use to users of SimEther. This includes exactly what aspects of Ethernet is included in the simulation.

5.1 Site states

Ethernet sites in SimEther are equipped with a transmitter, and receiver. Each of these contain states that model proper Ethernet sites. These include:

- Waiting - No action.
- Sense - Site transmitter senses the cable before transmitting.
- Send - Transmitter sends data on the cable.
- Back-off - Transmitter backs off after JAM signal sent.
- Jam - Transmitter sends JAM signal on collision detect.
- Receive - Receiver capturing data for this site.
- Monitor - Receiver capturing data for a different site. If this site is part of a bridge, it can then inform the bridge of an incoming frame.

Each site is equipped with a transmitter and receiver. Initially sites were modelled with as single multi-state variable. Later it was discovered this was insufficient. In particular, a network containing two sites, each wishing to transmit to the other is not able to be implemented with a single state variable. This type of transmission is required for the implementation of bridges.

5.2 Back-off method used

When a transmitting or receiving site detects a collision on the cable, the site will produce a jam signal, and then back-off for some amount of time. The Ethernet protocol uses a truncated binary exponential backoff algorithm to work out how long a transmitting site will wait before it attempts another transmission. SimEther uses a uniform random number distribution to calculate a back-off period. This difference with the original Ethernet protocol offers the advantage of smaller possible site retransmission delays. Since SimEther is a teaching tool, where the number of sites connected to a network is quite low, there would not be much to be gained by using the truncated binary exponential algorithm. The large back-off delays possible with this algorithm could mean long delays in the simulation between site state changes.

Having condoned the idea of truncated binary exponential back-off, sites have been equipped with a retry counter which would easily allow the algorithm to be installed in sites in the future. Real world testing would need to be done to see if an advantage would be gained in doing so.

5.3 Frame lengths

Frames created in SimEther are of constant length. While Ethernet frames can be of varying lengths, this feature was not included in the simulation. The reason for this is given by explaining what would happen if variable length frames were allowed.

If variable length frames, as chosen randomly by the application were allowed:

- The maximum allowable network size would not be obvious. Since the maximum network size allowed is half the length of a frame, different minimum frame sizes as depicted by SimEther would allow different size networks. A user trying to demonstrate an overly long network would not be able to produce a suitably short frame on demand.
- Having variable length frames could increase the simulation time dramatically. In Ethernet, the difference in allowable frame lengths is a factor of over 20. Clearly, trying to simulate events of such differing time magnitudes would be difficult to present to the user coherently.

If packet lengths were selected by the user there would be another problem. The simulation would not show the variation in packet lengths unless the user kept selecting different size frames.

Having the ability to select between constant length frames for demonstrating excessively long networks, and random length frames[1] for demonstrating standard network operation might solve these problems.

5.4 Screen snapshots to a file

A rudimentary screen snapshot feature has been implemented. It can be activated from anywhere within the program with the exception of any on-screen requesters. To create a screen dump, press the **F1** key. A tone will signify that the dump is in progress. Another higher pitch tone after a short while signifies the end of the dumping procedure. The resulting file resides in the current directory. The file is in the ppm (portable pixel map) file format. Multiple dumps from one program session will be labelled consecutively.

5.5 Default network load at startup

A default network is loaded upon startup of the program. By saving any network as **default.net**, you can have SimEther start up with any network you

[1] Random to some degree – perhaps not as variable as in Ethernet

choose.

5.6 Cable termination

Real Ethernet cable requires each end of the cable to be terminated with a 50Ω resistor. Failure to do so results in signal reflections.

In Ethernet, this reflection interferes with the source pulses being generated, causing errors. The terminators act to make signals behave as if there is no change in impedance at the cable ends - effectively making the cable seem an infinite length, and hence no reflection. Cable, in SimEther is automatically terminated. There is no need to terminate the ends of a bus to prevent reflections. Unfortunately nothing equivalent exists in the real world!

Chapter 6

Results

Looking back at initial outlines of the project, it is apparent that many of the original ideas and methodologies have made it to the final product.

SimEther is capable of demonstrating numerous interesting situations that might arise in Ethernet networks, including, but not limited to:

- A successful transmission between two sites
- Collision with possibly multiple back-off stages
- Errors due to excessive cable lengths
- Bridge learning
- Intelligent forwarding by bridges
- Buffering capability of bridges
- Locality of traffic

The amount that has been completed is pleasing, and SimEther is a useful tool. From past casual research, many other projects with associated applications appear to be incomplete, and hence only useful to other scholars continuing research in that field. It is believed that SimEther is at such a stage where it can be used by many casual users, and not just specialists. In particular, SimEther

has appeared to be stable and robust. This in itself will help it be accepted as a reliable tool for demonstration purposes.

In recent years the trend in usage of Ethernet has been away from the long bus winding over a floor of a building, or through an office, towards the use of hubs. By connecting users individually to a central hub, the system offers data security. With the hub decrypting the data for only the intended recipient, none of the other users have the chance to intercept data destined for another user.

Since hubs still use the Ethernet protocol, SimEther is still a valid tool for teaching purposes. It should just be kept in mind that the single, long network with many stations is now used less frequently than in the past.

Chapter 7

Looking to the future

Substantial progress has already been made in simulating Ethernet. There is not much more that could be done to illustrate Ethernet at its most basic level. The simulation could be made quantitative, but that would require substantial reworking of the program. It could be turned into a *minimax* style educational game where the end user has to maximise a users performance on the network, while minimising costs to build the network. This would involve substantially changing the program and would have a different target audience.

Automatic sites were suggested. Such sites run independently of the user, producing packets on their own accord. It is doubtful that such sites would be very useful in a pure teaching environment. To be able to demonstrate a point effectively, the demonstrator must have complete control over what happens in the Ethernet environment. A site which is producing spurious frames in one part of the network could interfere with a demonstration on another part of that network. Automatic sites would be more suited to quantitative network throughput simulation for example.

Backwards time control. While the time facility provided enables the user precise control in the forward temporal direction, there is no facility for stepping back in time. This could be useful if an event is accidentally skipped, or if a student wished to see a particular event again.

Regarding the user interface, there are numerous things that have not been completed to my satisfaction:

- A colour selector could be implemented to enable SimEther to be used on grey-scale displays – for example notebook machines.

- A clear all feature for completely erasing the network would be useful. It is possible to load a blank network to get around this problem, but this is a kludge. Also some way to clear the information windows. When a user loads a new network, the old information remaining in the windows is no longer needed, and is in fact a hindrance.

- Pull down menus. All functions are currently *out in the open*, ready to be clicked upon. With the increasing number of functions, some form of menu is required. Either pop up or pull down, for functions such as loading, saving and quitting would be very beneficial.

- The present file requester is of bare-bones design. There is a surprising amount of work involved in designing and implementing a truly productive file requester. The current system only allows selection of networks by file name. It is therefore important to save networks with meaningful names. A file requestor where in addition to network descriptions (given by name), a miniature diagram of the saved network is also drawn would be better. This would let the user see at a glance what they would be loading, rather than the current system which relies on sensible file names being used. Time was a limitation on the implementation however, and this was one of the (few) things that were omitted.

One aspect not looked at was the evaluation of SimEther in actual use. This was considered beyond the scope of the project.

Chapter 8

Bibliography

Bibliography

[1] ADAMS, L. *Supercharged C++ Graphics.* Windcrest/McGraw-Hill, 1992.

[2] CUSHMAN, W. H., AND ROSENBERG, D. J. *Human factors in product design.* Elsevier, 1991.

[3] DEMEL, J. T., AND MILLER, M. J. *Introduction to Computer Graphics.* Brooks/Cole Engineering Division, 1984.

[4] EZZELL, B. *Grphics Programming in Turbo C++.* Addison-Wesley, 1991.

[5] KERNIGHAN, B. W., AND RITCHIE, D. M. *The C Programming Language.* Prentice Hall, 1988.

[6] SHNEIDERMAN, B. *Designing the user interface.* Addison-Wesley, 1987.

[7] SILVERSTEIN, L. D. Human factors for color display systems. Chapter found in book called Color and the computer.

[8] SNAPPER, B., AND FISH, F. Fish prices on the European food market. *What Fish? (Not to be confused with What Hi-Fi? magazine)* (Oktober 1993), 42. Probably not to be taken seriously.

Appendix A

Installing SimEther

The supplied diskette contains both source code for SimEther, and also a distribution archive. The distribution archive contains the SimEther executable, numerous demonstration networks, and a **readme.txt** file containing version information.

Installation of the executable code is reasonably straightforward. When you have decided what directory you wish to install the code to, move to that directory. From this directory, run the distribution archive, which will then self extract into your current directory.

For example, say you wish to install SimEther onto a fixed disk, in its own subdirectory. First create the subdirectory on the fixed disk with the **mkdir** command. Then move into the directory you have just created using the **cd** command. Now run distribution archive. To do this, if the floppy disk containing the archive is in drive **A:**, enter the command **A:\sether.exe**. The files will be extracted into the current directory.

To start SimEther, enter **se**.

Appendix B

Tutorial

At this point it is assumed that you have SimEther extracted into the current directory. If you have not, see the appendix on installation.

Two tutorials follow. The first deals with the basics of SimEther, while the second demonstrates some of the more in-depth features.

B.1 First Contact

Start SimEther by entering **se**.

You should be greeted with a grey-scale desktop looking something similar to figure B.1 If you are not, then something has gone wrong. The two most common problems are that SimEther is not being run from the directory it was extracted to, or that a suitable mouse driver is not installed. You will receive a message warning you of a missing mouse driver, which you can then install before running the program again. Failures relating to missing graphics drivers, or fonts, can be handled by reinstalling the code. See the installation appendix.

If you have reached the point where a mouse cursor appears on the screen, and it responds to mouse movement, things should be straightforward from this point on.

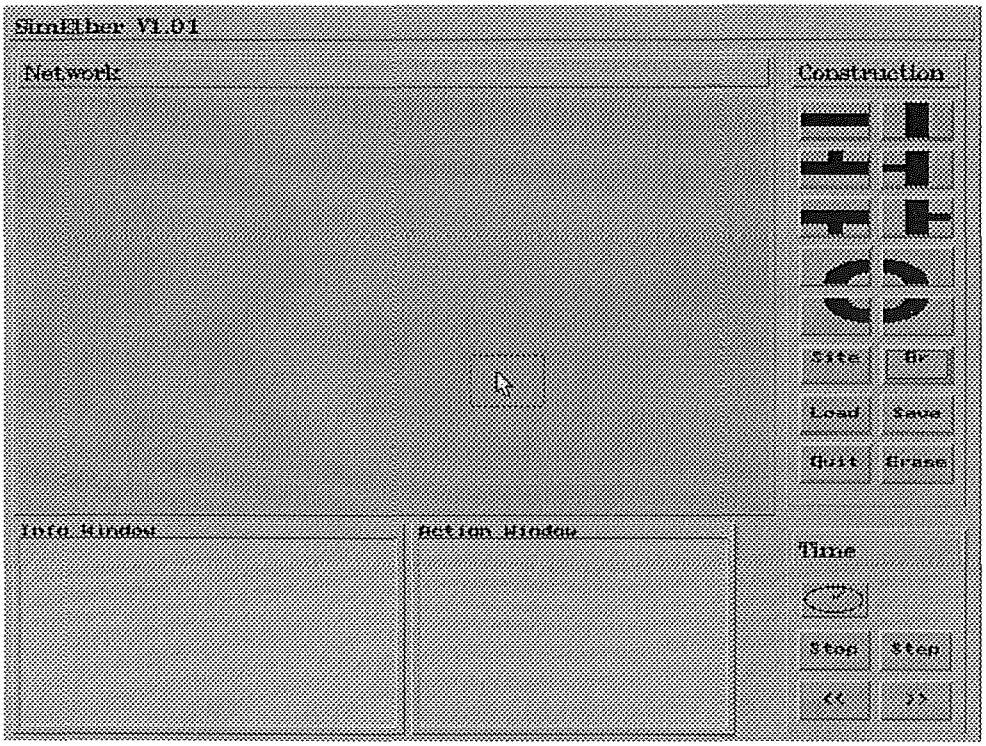

Figure B.1: SimEther straight after starting up

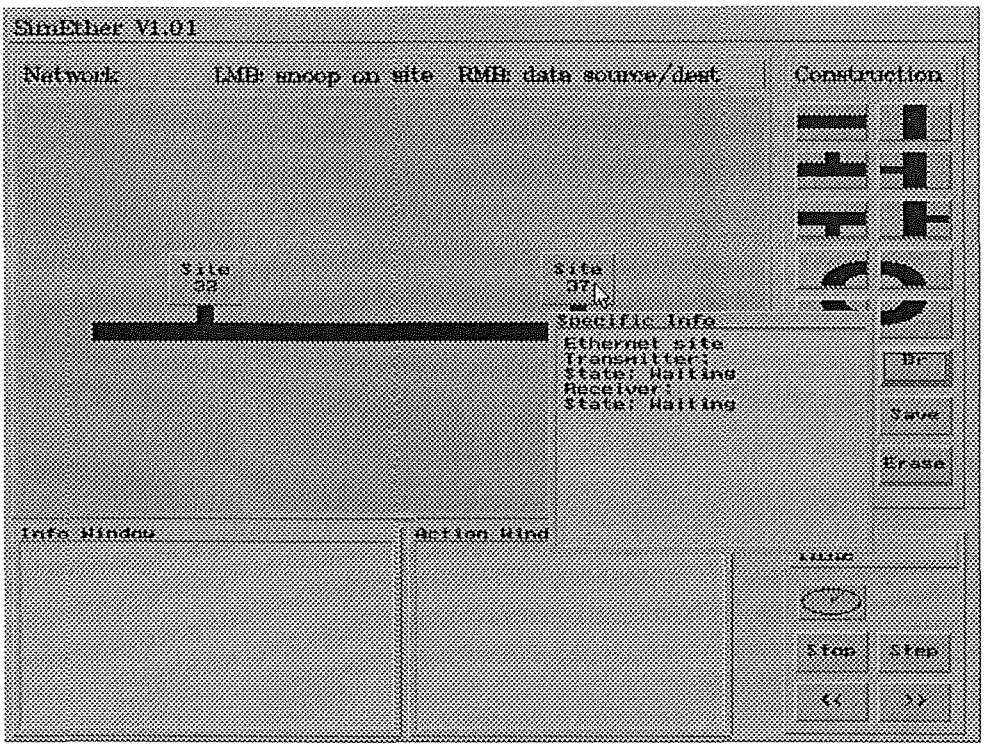

Figure B.2: A simple network

Try moving the arrow cursor over various items on the desktop. You will be informed of what actions will take place, should you depress one of the mouse buttons, at the top right hand corner of the network window. Most actions run off the left mouse button (LMB). Try selecting a piece of cable, and then placing it in the network window. Try selecting a site, or a bridge and placing that on the network too. For sites and bridges to work properly, they must be connected to cable *taps*, and not just placed at the end of busses, or ends of pieces of cable. A cable tap is the small piece of cable coming off at right-angles to a straight piece of cable.

Sites should be connected to one piece of cable at most. If a site is connected to more than one, it will chose a single arbitrary piece to send and receive on.

Building networks is quick once you get the hang of it. For now, try loading a new network. Select **load**, and enter **simple.net**. A basic network containing two sites will be loaded. Notice how the cable is used to build a network?

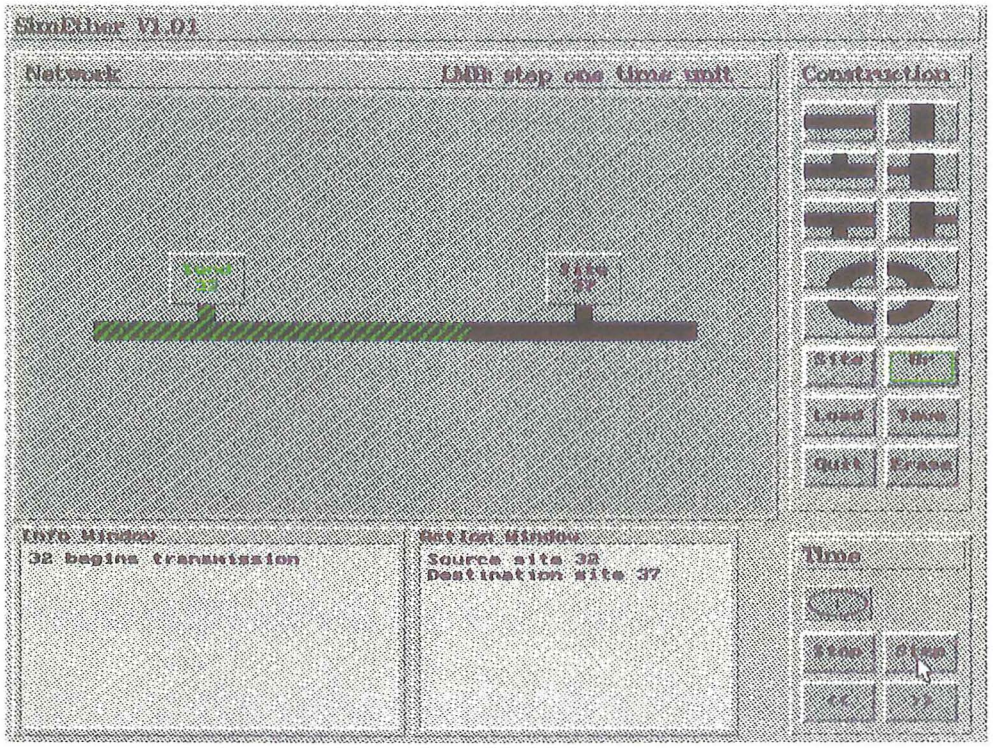

Figure B.3: The start of a transmission

Moving the cursor over the network reveals that the left mouse button can be used for examining anything you wish. Try it out. See figure B.2. Click a mouse button to clear the info window.

Now try sending a frame from one site to another. Move to the left site, and click the right mouse button. The action window will reveal that the left site is set to transmit. Now click on the right hand site with the right mouse button[1]. Now advance time to see the result. To do this, click the **step** button in the time window a few times. Notice the clock in the time window advance? See figure B.3. See the frame flow out of the left site, and along the network? The info window in the lower left corner of the screen lets you know what has recently happened on the network.

It is possible to examine the data on the cable by clicking with the LMB at

[1] From now on know as the RMB

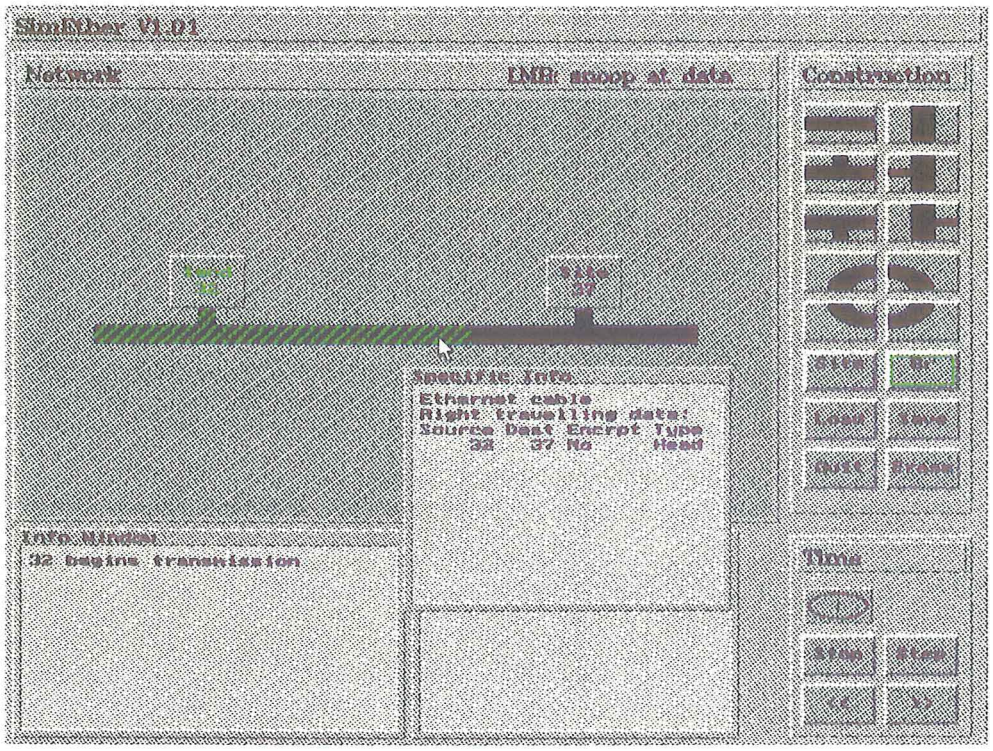

Figure B.4: Spying on the cable!

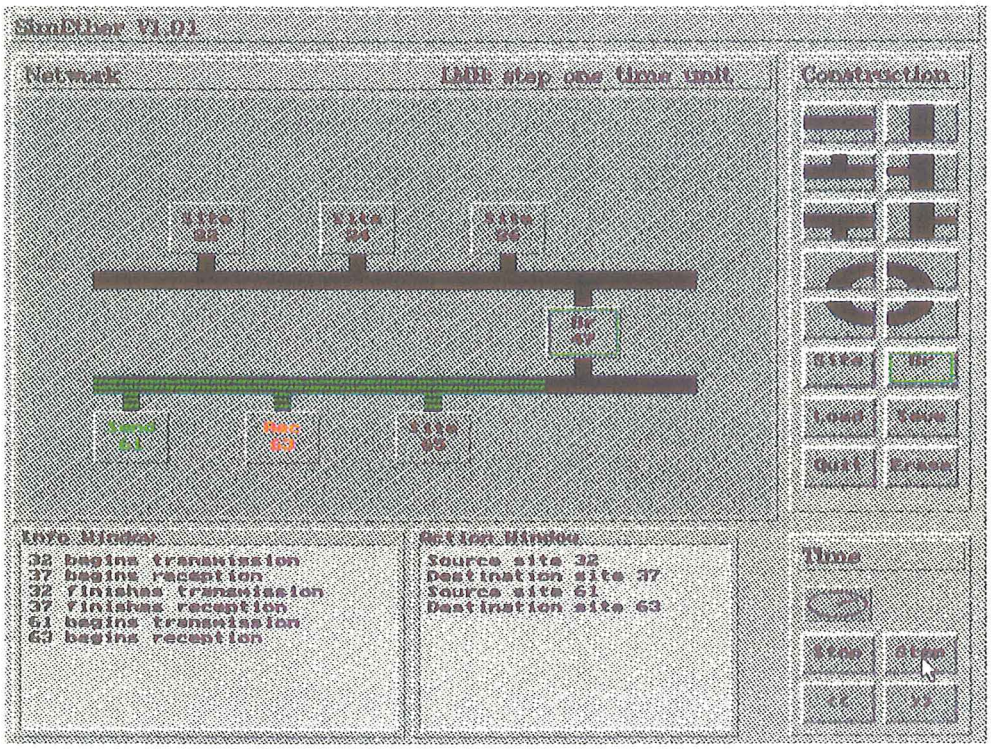

Figure B.5: Frame being received

various positions on the cable. See figure B.4.

Step the time some more, and you will see the frame flow further along the network, and the right hand site will begin to receive the frame. Step some more until the frame is finished being transmitted, and flows off of the network.

This concludes the first tutorial.

B.2 Abridged Network Layout

Load the network **bridge.net**. You will be greeted with two networks joined by an Ethernet bridge.

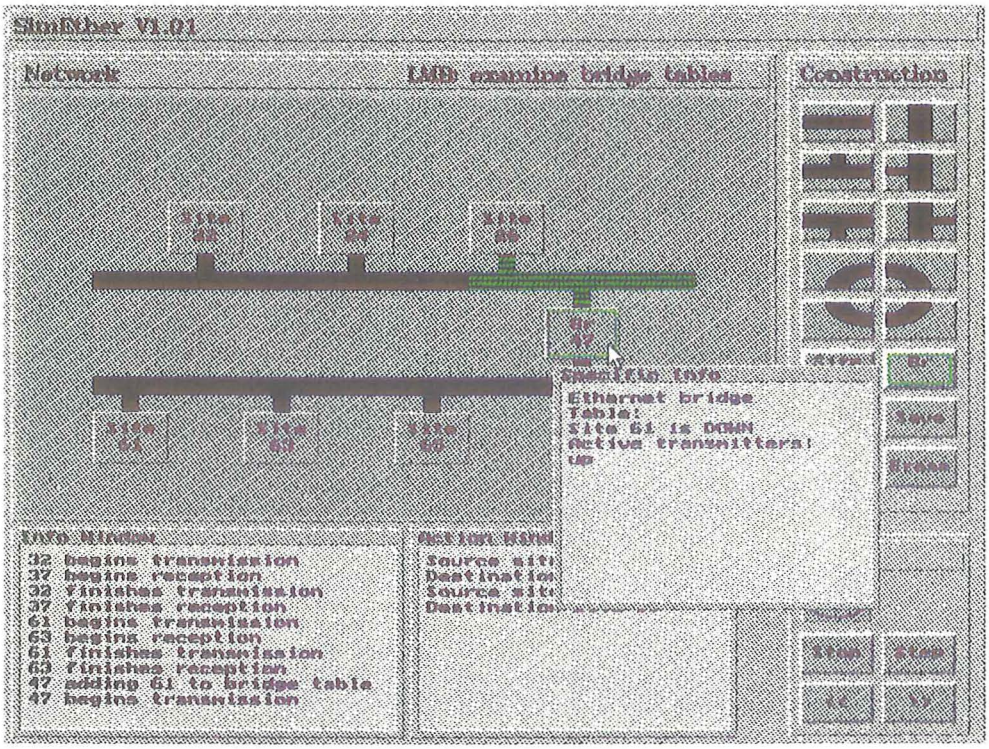

Figure B.6: Examining the bridge

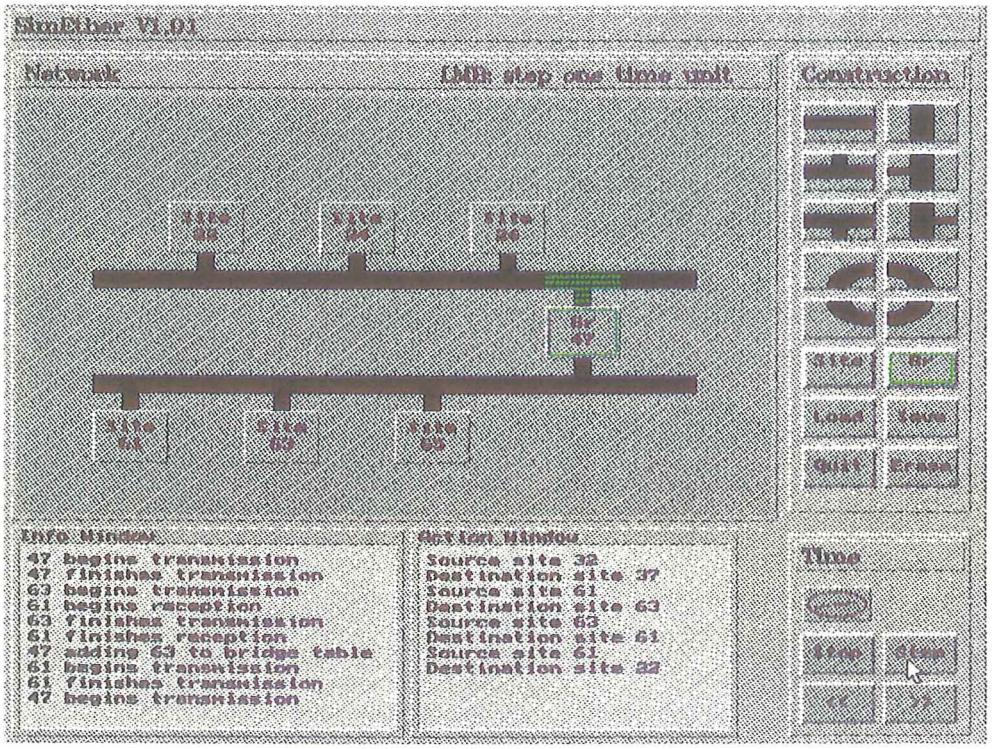

Figure B.7: Bridge starting transmission

Transmit a frame from site 61, to site 63. Step the time (see figure B.5) until the frame flows off the bottom network. After this has happened, the bridge will begin to transmit onto the upper network. This is done because the bridge does not know where site 63 is, and will forward all messages it receives for that site, to any other connections it may have. In this case it just forwards the frame to the top network. Examining the bridge reveals what is going on. See figure B.6. Site 61 is registered with the bridge as being connected to the lower connection. This information was picked up from the transmission from site 61 to site 63. From this point on, any frames destined for site 61 that are received by the bridge, will be forwarded downwards. Also, the bridge now knows not to forward frames to site 61 upwards, as this would lead to pointless traffic.

Advance the time some more until the top frame flows off the network. Now transmit from site 63 to 61. Step the time, and notice how the bridge does not forward the frame.

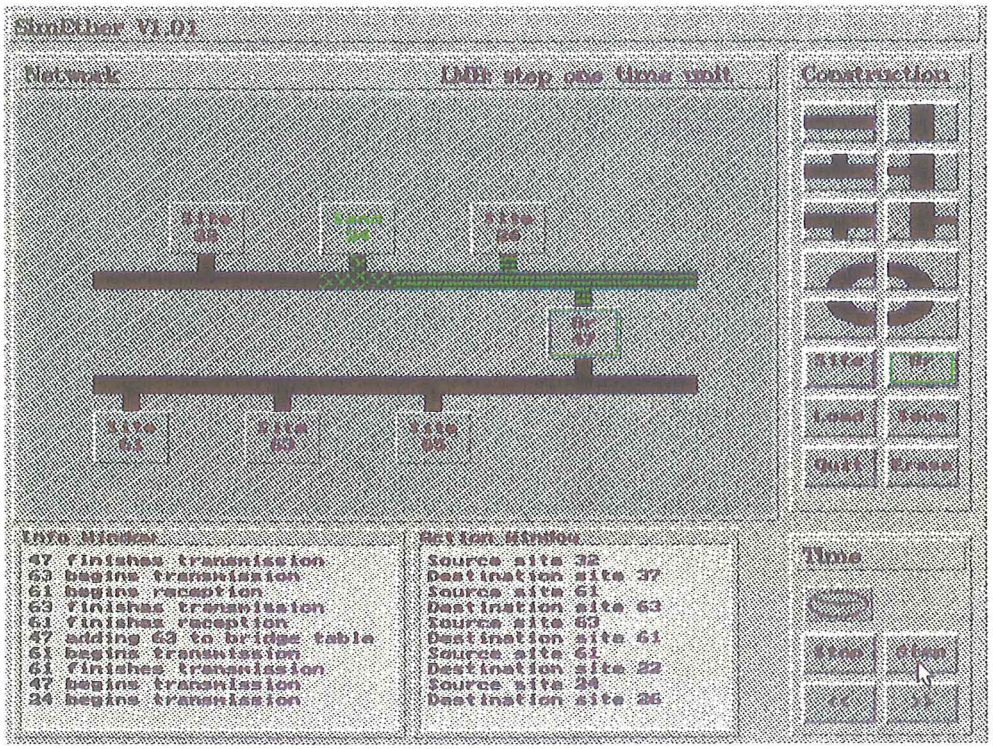

Figure B.8: A collision about to occur

To illustrate that bridges are a number of Ethernet sites connected together with buffering mechanisms, try the following. Send from site 61 to site 22. Step the time, until the bridge receives the whole frame, and then begins to forward it. See figure B.7. At this point the transmission on the bottom network is deemed successful, even if a problem occurs on the top network.

Try sending from site 24 to site 26. Step time. See figure B.8. A collision occurs, and a back-off procedure is applied. See figure B.9. Step time some more, and see that after a while one site will wait for the other to transmit. Site 22 receives the data from 61. See figure B.10. Step some more and you will see site 24 begin to send when the cable is free. See figure B.11. Lastly, since site 26 is not known by the bridge, it will forward the frame to the bottom network. See figure B.12.

This concludes the second tutorial.

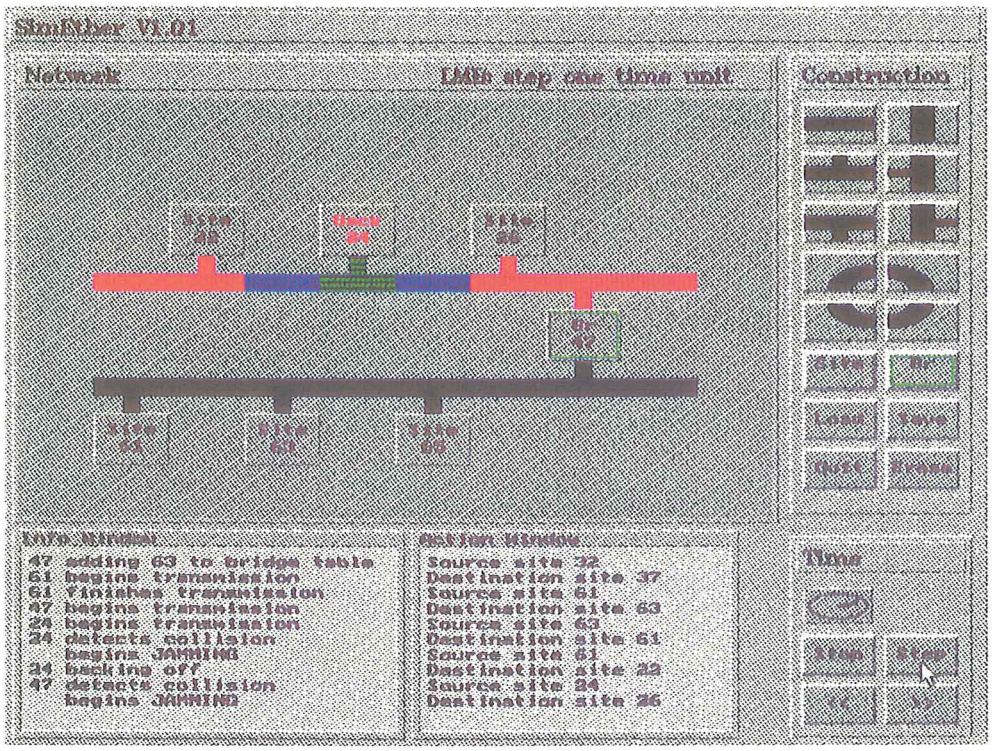

Figure B.9: Bridge frame and site frame colliding

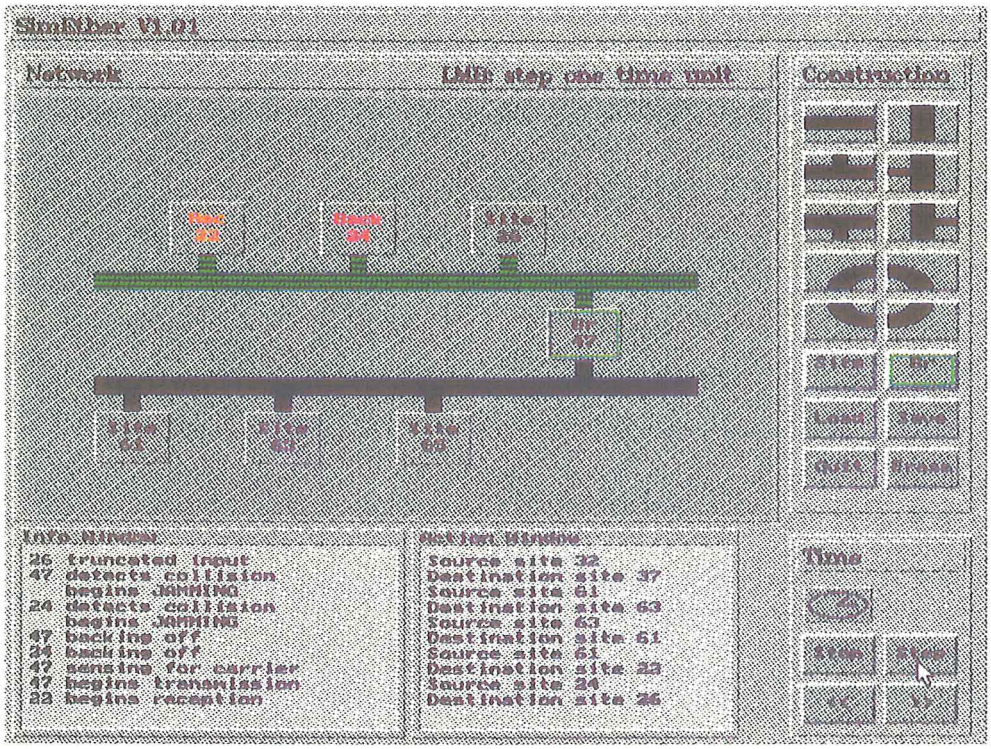

Figure B.10: Bridge retransmission successful

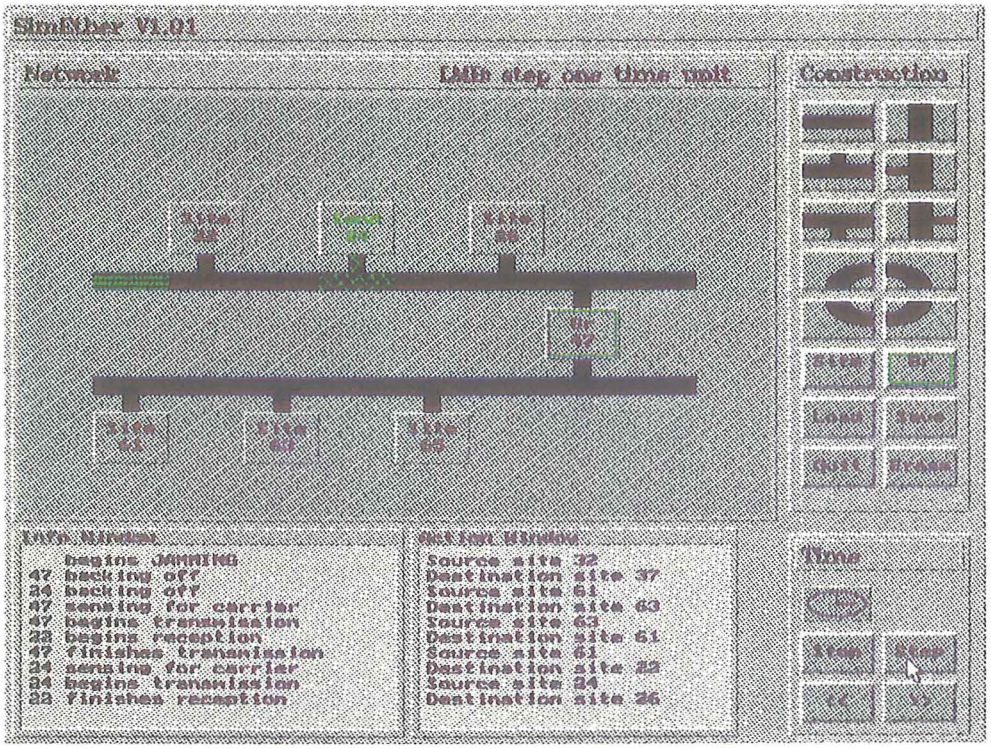

Figure B.11: Site transmitting on free cable

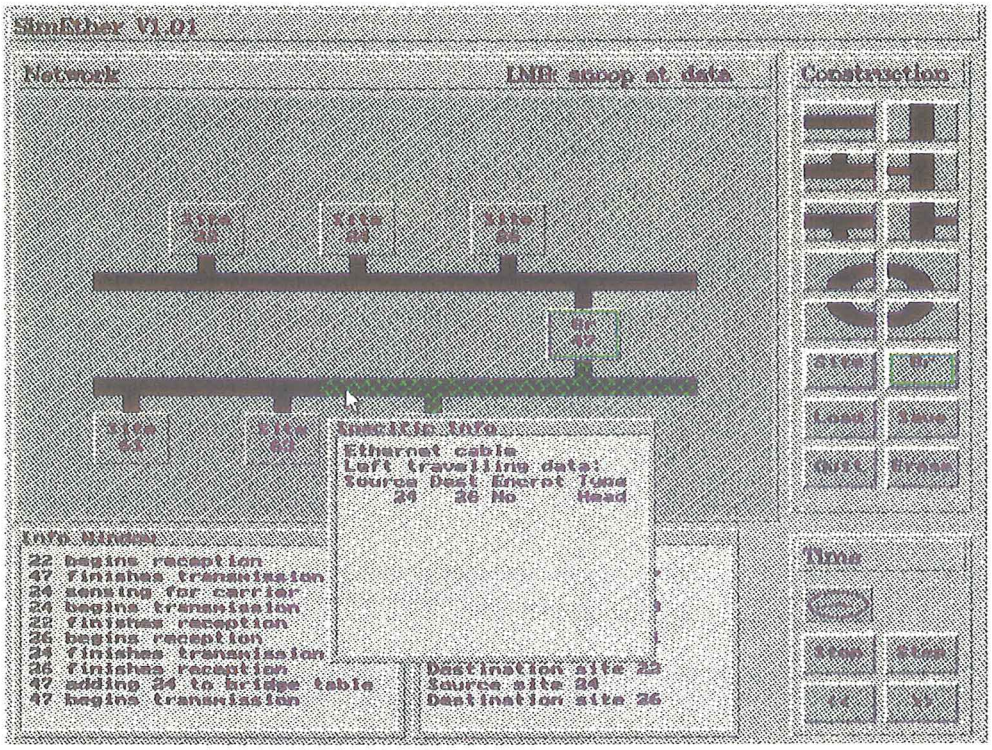

Figure B.12: Bridge forwarding frame to lower bus

Appendix C

Functions

What follows is a breakdown of all the functions involved in the construction of SimEther. Also disclosed are all the structures used, and all defined values. Functions are listed file by file – the files in alphabetical order. There is no particular ordering for the functions within a file.

It is intended that anybody wishing to alter SimEther be able to use these function definitions as a guide to navigating their way round the program.

C.1 colour.h – Contains platform dependent colour routines

void set_colours();
Selects the colour palette for a graphics screen, and sets the colour values to suitable colours

C.2 display.h – Contains all screen output routines

void draw_network(netelement net[NETHEIGHT][NETWIDTH], struct icon *icons);

Takes a network representation, and all the screen icons. Draws the contents of each network element at its correct location.

void draw_nothing(int xloc, int yloc, int xsize, int ysize);
Draws a blank rectangle with the passed dimensions.

void draw_nothing_text(int xloc, int yloc, int xsize, int ysize, char *text);
Draws a blank rectangle, but additionally prints some text in it.

void draw_cable(int xloc, int yloc, int xsize, int ysize, int type, int colour, int fillstyle);
Draws a piece of cable of the screen.

void draw_site(int xloc, int yloc, int xsize, int ysize, int label, int state);
Draws a site on the screen, with a label. The state of the site is also passed so extra highlighting can be applied.

void draw_bridge(int xloc, int yloc, int xsize, int ysize, int label, int state);
Draws a bridge on the screen.

void window_work(int outerleft, int outertop, int xsize, int ysize, char *text);
Draws a purely decorative window on the screen, with a title.

struct icon *do_icons(struct icon *icons, int iconleft, int nettop);
Initialises, and draws all the icons on the screen.

void display_info(netelement net[NETHEIGHT][NETWIDTH], struct icon *the-icon);
Displays on the screen, detailed information about the passed network item.

int request_bool(char *question);
Draws a boolean requester, and returns the users reply to the question, which is displayed in the requester.

int request_unary(char *statement);
Draws a requester, and prints a statement in it. The user is required to make an acknowledgement before the routine returns.

void toggle_net_icon(struct icon *over);
Toggles the displayed state of an icon, from selected to not selected, and vice

versa.

char *request_file();
Displays a file requester, gets a selection from the user, and returns with the selected file name.

char *read_string(int left, int top);
Reads a string of characters from the keyboard, while echoing them to the screen at the given absolute locations. Returns when the return key is pressed, returning the string.

C.3 file.h - Contains file access routines

```
struct text_list
{
    char *name;
    struct text_list *next;
};
```

The structure in which directory listings are produced. Essentially a linked list of names.

int save_network(netelement[NETHEIGHT][NETWIDTH], char *filename);
Saves the passed network into a file with the passed file name.

int load_network(netelement net[NETHEIGHT][NETWIDTH], char *filename);
Loads the network, in the file with the passed file name, into the passed network.

struct text_list *file_directory(void);
Produces a list of files in the current directory with the extension ".net".

C.4 icon.h - Contains all icon routines

```
struct icon
```

```
{
    struct icon *next;
    int left, top, right, bottom;
    int id;
    int type;
    int state;
    int aux;
};
```

Structure for keeping track of icons on the screen. Essentially a linked list of icon data.

struct icon *icon_find(struct icon *icons, int x, int y);
Returns a pointer to an icon that is at the passed screen location.

struct icon *icon_add(struct icon *icons, int left, int top, int right, int bottom, int id, int type, int state, int aux);
Add a new icon to the list of current icons.

void icon_highlight(struct icon *theicon);
Highlights an icon for easier viewing.

void icon_outline(struct icon *theicon);
Produces an outline around an icon, for easier viewing.

void icon_outline_all(struct icon *icons);
Outlines every one of a list of icons.

struct icon *find_net_icon(struct icon *icons, int i, int j);
Finds the icon belonging to a particular network element.

C.5 list.h - Contains list implementation routines

```
typedef struct
{
    int x;
    int y;
```

```
} location;

typedef struct
{
    location source;
    location dest;
    int encrypted;
    int type;
} data_packet;
```

The structure for describing a segment of a frame.

```
struct list_element
{
    data_packet *value;
    struct list_element *next;
};
typedef struct list_element *listptr;
```

The structure for keeping track of a number of frame segments. Essentially a linked list.

listptr list_add(listptr current, data_packet *newelement);
Add a data packet to a list of data packets. Returns the new list.

int list_size(listptr head);
Returns the number of data packets in the data packet list

data_packet* list_return(listptr head, int index);
Returns a pointer to a particular data packet. The list is indexed from zero upwards.

listptr list_kill(listptr head, int killdata);
Deletes the whole list and frees up the memory used by the list. If killdata is nonzero, the user data pointed to by the list will also be freed up.

C.6 se.h - Contains SimEther constants

```
#define NETWIDTH 10 /* Width of the network in blocks */
#define NETHEIGHT 8 /* Height */


#define PACKET_LENGTH 24 /* Length in segments (blocks), of a packet */


/* Type of equipment located at a particular point on the net */


#define EQUIP_NONE 0
#define EQUIP_SITE 1
#define EQUIP_CABLE 2
#define EQUIP_BRIDGE 3


/* Propagation directions of sites (must be able to be 'anded') */


#define UP 1
#define RIGHT 2
#define DOWN 4
#define LEFT 8


/* Cable ID types */


#define HORTTAP 1
#define HORBTAP 2
#define VERLTAP 4
#define VERRTAP 8
#define HOR 16
#define VER 32
#define LB 1024
#define LT 128
#define RB 256
```

```
#define RT 512

/* Other object IDs in general */

#define NOTHING 64 /* An empty space */
#define SITE 42
#define BRIDGE 43

/* IDs for the momentary icons */

#define TIME_FASTER 65
#define TIME_SLOWER 66
#define TIME_STOP 67
#define TIME_STEP 68
#define LOAD 44
#define SAVE 45
#define QUIT 99

/* Data type */

#define HEAD 0
#define MIDDLE 1
#define TAIL 2
#define JAM 3

/* Site states */

#define STATE_SEND 0
#define STATE_BACKOFF 1
#define STATE_JAM 2
#define STATE_WAIT 3
#define STATE_SENSE 4
```

```
#define STATE_RECEIVE 6
#define STATE_MONITOR 7
#define STATE_ICON 8

/* What we can detect by looking at a piece of cable */

#define SENSE_NOTHING 0
#define SENSE_DATA 1
#define SENSE_COLLISION 2
#define SENSE_JAM 3

/* The two parts of an ethernet site */

#define TRANSMITTER 1
#define RECEIVER 2

/* The types of icons that exist */

#define BUTTON 1
#define NETELEMENT 2
#define STACK 3
#define MOMENT 4

/* All the colours we can use */

#define COLOUR_TEXT 0
#define COLOUR_NOTHING 0
#define COLOUR_PACKET 1
#define COLOUR_PACKETENC 3
#define COLOUR_PACKETJAM 2
#define COLOUR_PACKETCOLL 4
#define COLOUR_SHADOW 9
```

```
#define COLOUR_BUTTON 11
#define COLOUR_DESKTOP 12
#define COLOUR_TEXTWINDOW 13
#define COLOUR_SUN 14
#define COLOUR_HIGHLIGHT 2


/* Data encryption */


#define ENCRYPTED 1
#define NOT_ENCRYPTED 0


/* Boolean requester uses YES/NO */


#define NO 0
#define YES 1


#define PRESSED 1
#define NOTPRESSED 0


/* Error return codes */


#define OK 0
#define ERROR -1
```

C.7 serout.h - Contains general simulation routines

```
struct table
{
```

```
    int direction;
    int site;
    struct table *next;
};
typedef struct table table;
```

A structure for keeping a list of site numbers, and the direction they are in. Designed primarily for bridge tables, so that bridges can keep track of what sites are where.

```
struct bridge_buf
{
    location source;
    location dest;
    int encrypted;
    struct bridge_buf *next;
};
typedef struct bridge_buf bridge_buf;
```

A structure for maintaining a buffer of frames to send from a site.

```
typedef struct
{
    int type;
    listptr leftdir;
    listptr rightdir;
    listptr updir;
    listptr downdir;
} cable_segment;
```

A structure for describing a piece of cable. Cables can contain data. In this case, pointers to lists of frame segments that propagate along the wire.

```
typedef struct
{
```

```
    int connections;
    int state;
    int countdown;
    data_packet *input;
    location dest;
    location source;
    int encrypted;
    int retry;
    int automatic;
    int xloc;
    int yloc;
    int last;
} net_site;
```

A structure for describing a network site. Describes either an Ethernet transmitter, or a receiver.

```
typedef struct
{
    bridge_buf *up, *down, *left, *right;
} bridge_buffers;
```

A structure for describing the buffers a bridge has. In this implementation, a bridge has a buffer for each Ethernet connection.

```
typedef struct
{
    net_site *left_t, *right_t, *up_t, *down_t, *left_r, *right_r,
    table* bridge_table;
    bridge_buffers outqueue;
} net_bridge;
```

A structure that describes a bridge. bridges are made up of four Ethernet units, each consisting of a transmitter, and a receiver. Bridges also have tables to facilitate frame forwarding, and queues for buffering outgoing frames.

```
union equipment
{
    net_bridge *bridge;
    net_site* site; /* Points to an Ethernet site */
    cable_segment* cable; /* Points to a piece of cable */
};
```

A piece of equipment is either a bridge, a site, or a piece of cable.

```
typedef struct
{
    int equipment;
    union equipment current;
    union equipment temp;
} netelement;
```

Each position of the network has an equipment type, as well as two possible instances of that piece of equipment.

int snapshot_screen(void);
Bare bones screen snapshot routine. Saves snapshots to the current directory, in PPM picture format.

void free_site(net_site *site);
Frees up all memory used by a site.

void free_cable(cable_segment *cable);
Frees up all memory used by a piece of cable, including any data currently on this piece of cable.

void free_bridge(net_bridge *bridge);
Frees up all memory used by this bridge.

void free_net(netelement net[NETHEIGHT][NETWIDTH]);
Frees up all the memory used by a network. The equipment type of each element on the network is set to empty.

char *s_dup(char *);
Duplicates a string. That is, allocates memory, and then makes a copy.

long mem_avail();
Debugging routine for testing to see how much memory is available for allocation. Used mainly to find memory leakages.

bridge_buf *bridge_buf_add(bridge_buf *buffer, int srcx, int srcy, int destx, int desty, int enc);
Add a frame to a bridge transmission buffer (FIFO). Return the resulting buffer.

bridge_buf *bridge_buf_delete(bridge_buf *buffer);
Remove the frame at the head of the list. Return the resulting frame.

int maintain_ethernet(netelement net[NETHEIGHT][NETWIDTH], struct icon *icons, int xloc, int yloc);
Does all the housekeeping for an Ethernet device. This involves looking after bridges, and calling site maintenance routines.

int maintain_site(netelement net[NETHEIGHT][NETWIDTH], net_site *site, int xloc, int yloc, int mode);
Maintains an Ethernet site. Housekeeping routine. This involves for example, sites transmitting onto a piece of cable, sensing for collisions etc. . .

table *table_delete(table *atable, int sitenum);
Deletes a site entry from a bridge table. Returns the resulting table.

table *table_add(table *atable, int sitenum, int direction);
Adds site information to the bridge table. Returns resulting bridge table.

int table_find(table *atable, int sitenum);
Searches the bridge table for a particular site. Returns the direction the site is connected at, should this information be in the bridge table.

void bridge_action(netelement net[NETHEIGHT][NETWIDTH], net_bridge *this_bridge, int from);
Routine that does bridge housekeeping. For example, takes an incoming frame and updates the bridge tables. Also takes frames off the outgoing queue, and starts outward transmission.

int init_net(netelement net[NETHEIGHT][NETWIDTH]);
Sets the entire network to contain empty spaces.

net_site *create_site(int xloc, int yloc, int direction);
Allocates and initialises an Ethernet site.

net_bridge *create_bridge(int x_loc, int y_loc);
Allocates and initialises an Ethernet bridge.

int check_site(netelement net[NETHEIGHT][NETWIDTH], int x, int y);
Performs a rough sanity check on a site.

cable_segment *create_cable(int type);
Allocates and initialises an piece of cable of the passed type.

int check_net(netelement net[NETHEIGHT][NETWIDTH]);
Does a rough sanity check on the passed network.

data_packet *dup_data(data_packet *source);
Duplicates a frame segment.

void propagate(netelement net[NETHEIGHT][NETWIDTH], int yloc, int xloc, listptr packetlist, int direction);
Propagates data on the cables in the network.

int site_connection(netelement net[NETHEIGHT][NETWIDTH], int xloc, int yloc);
Returns the direction of connection of a site at the described location of the passed network.

cable_segment *find_cable(netelement net[NETHEIGHT][NETWIDTH], int xloc, int yloc, int connections);
Returns a pointer to the piece of cable the passed site is connected to.

int check_transmission(netelement net[NETHEIGHT][NETWIDTH], int xsrc, int ysrc, int xdest, int ydest);
Checks the validity of a transmission request. For example a transmission to a non-existent site is illegal.

int transmit(netelement net[NETHEIGHT][NETWIDTH], net_site *this_site);
Transmits frame segments from the passed site onto the adjoining piece of cable.

int sense_cable(cable_segment *cable);
Describes the state of affairs on the passed piece of cable. For example, whether the cable is clear, or busy etc.

void update_net(netelement net[NETHEIGHT][NETWIDTH], struct icon *icons);
The routine for network housekeeping. Updates all Ethernet devices, and propagates all signals on the network.

int site_transmit(netelement net[NETHEIGHT][NETWIDTH], net_site *this_site, int sourcex, int sourcey, int destx, int desty);
Sets up transmission from one site to another. Doesn't actually do the transmission, but puts the transmitting site into the required state.

void fill_mem(int space);
Debugging routine. Used to allocate approximately all available memory except for the passed number of bytes. Used to test for example, the behaviour of routines with insufficient free memory.

char *describe_data(data_packet *data, char *junk);
Routine that produces a string containing useful information about the passed frame segment

char *describe_mouse_actions(netelement net[NETHEIGHT][NETWIDTH], struct icon *over, struct icon *currentbutton);
Routine that depending on what the mouse cursor is over, and what icons are selected, will return a string describing what mouse button actions will have what effect

int fill_style(cable_segment *cable);
Given a pointer to a piece of cable, will return a suitable style for display on the screen. For example, a collision will be represented by solid fill.

int random_num(int modnum);
Produces a rough pseudo-random number in the range 0->modnum-1.

C.8 clock.hpp - Platform dependent clock object

void init(int, int, int, int);
Draw an initial picture of the clock, on the screen.

void update();
Increment the time by some small value.

void display();
Draw the clock with the current time.

void draw_hands(int minutes, int seconds, int colour);
Draw just the hands of the clock, in the passed colour.

C.9 request.hpp - Platform dependent requester object

int init(int left_of_win, int top_of_win, int x_size, int y_size);
Saves the background information, and draws a blank space on which gadgets, or other temporary items can be placed.

int bordersize();
Returns the size of the border at the right hand, and bottom side of the requester. This should not be written into as it contains the requesters shadow.

void close();
Restores the requester to whatever was there beforehand.

C.10 window.hpp - Platform dependent text window

void init(int left, int top, int x_size, int y_size, char *title);
Draws the window with the passed location, size and title.

void clear();
Does a clear screen on the window. Not currently implemented.

void display(char *);
Displays the passed string on the window.

Appendix D

Source code

D.1 colour.h

```
void set_colours();
/* Selects the colour palette for a graphics screen, and sets the
colour values to suitable colours */
```

D.2 colour.c

```
void set_colours()
{
  int i, x;

  for (i = 0; i <= 15; i++)
    setpalette(i, i);

  /* Values are in the range (0-63) */

  setrgbpalette(0, 0, 0, 0);
  setrgbpalette(1, 0, 35, 0);
  setrgbpalette(2, 0, 15, 50);
  setrgbpalette(3, 60, 30, 0);
  setrgbpalette(4, 50, 0, 0);
  for (i = 5; i <= 15; i++)
```

```
  {
    x = (i - 4) * 5.72;
    setrgbpalette(i, x, x, x);
  }
}
```

D.3 display.h

```
#include <conio.h>

void draw_network(netelement net[NETHEIGHT][NETWIDTH], struct icon *icons);
/* Takes a network representation, and all the screen icons.  Draws
the contents of each network element ar its correct location. */

void draw_nothing(int xloc, int yloc, int xsize, int ysize);
/* Draws a blank rectangle with the passed dimensions */

void draw_nothing_text(int xloc, int yloc, int xsize, int ysize, char *text);
/* Draws a blank rectangle, but additionally prints some text in it */

void draw_cable(int xloc, int yloc, int xsize, int ysize, int type,
                int colour, int fillstyle);
/* Draws a piece of cable of the screen */

void draw_site(int xloc, int yloc, int xsize, int ysize, int label,
               int state);
/* Draws a site on the screen, with a label.  The state of the site
is also passed so extra highlighting can be applied. */

void draw_bridge(int xloc, int yloc, int xsize, int ysize, int label,
                 int state);
/* Draws a bridge on the screen */

void window_work(int outerleft, int outertop, int xsize, int ysize,
                 char *text);
/* Draws a purely decorative window on the screen, with a title */

struct icon *do_icons(struct icon *icons, int iconleft, int nettop);
/* Initialises, and draws all the icons on the screen */

void display_info(netelement net[NETHEIGHT][NETWIDTH], struct icon *theicon);
/* Displays on the screen, detailed information about the passed
network item */
```

```
int request_bool(char *question);
/* Draws a boolean requestor, and returns the users reply to the
question, which is displayed in the requestor */

int request_unary(char *statement);
/* Draws a requestor, and prints a statement in it.  The user
is required to make an acknowledgement before the routine returns. */

void toggle_net_icon(struct icon *over);
/* Toggles the displayed state of an icon, from selected to not
selected, and vica versa */

char *request_file();
/* Displays a file requestor, gets a selection from the user, and
returns with the selected file name */

char *read_string(int left, int top);
/* Reads a string of characters from the keyboard, while echoing
them to the screen at the given absolute locations.  Returns
when the return key is pressed, returning the string. */
```

D.4 display.c

```
char *read_string(int left, int top)
{
  char *text;
  int current = -1;
  int this_char;
  int max_string_length = 16; /* Maximum number of printable chars */

  if ((text = (char *)malloc((max_string_length+1)*sizeof(char))) == NULL)
    return NULL;

  text[0] = '\0';

  /* While string not too long, and key not return key */
  while (((this_char = getch()) != 13) && (current < max_string_length-1))
  {
    setcolor(COLOUR_DESKTOP); /* Change */
    outtextxy(left, top, text);

    if (this_char == 8) /* Backspace char */
```

```
    {
      if (current > -1)
      {
        current--;
        text[current+1] = '\0';
      }
    }
    else /* Printable char */
    {
      current++;
      text[current] = this_char;
      text[current+1] = '\0';
    }

    setcolor(COLOUR_TEXT);
    outtextxy(left, top, text);
  }

  return text;
}


void draw_network(netelement net[NETHEIGHT][NETWIDTH], struct icon *icons)
{
  int i, j;
  struct icon *current;

  for (j = 0; j < NETHEIGHT; j++)
    for (i = 0; i < NETWIDTH; i++)
    {
      current = find_net_icon(icons, i, j);
      if (net[j][i].equipment == EQUIP_NONE)
      {
        draw_nothing(current->left, current->top,
                     current->right-current->left+1,
                     current->bottom-current->top+1);
      }
      else
      if (net[j][i].equipment == EQUIP_SITE)
      {
        draw_site(current->left, current->top,
                  current->right-current->left+1,
                  current->bottom-current->top+1,
                  current->state+current->aux*NETWIDTH,
                  net[j][i].current.site->state);
      }
      else
      if (net[j][i].equipment == EQUIP_CABLE)
```

```
      {
        draw_cable(current->left,
                   current->top,
                   current->right-current->left+1,
                   current->bottom-current->top+1,
                   net[j][i].current.cable->type,
                   BLACK,
                   SOLID_FILL);
      }
      else
      if (net[j][i].equipment == EQUIP_BRIDGE)
      {
        draw_bridge(current->left,
                    current->top,
                    current->right-current->left+1,
                    current->bottom-current->top+1,
                    current->state+current->aux*NETWIDTH,
                    STATE_WAIT);
      }
      else
        printf("Don't know how to display this!\n");
    }
}


void draw_nothing(int xloc, int yloc, int xsize, int ysize)
/* Routine to draw a 'nothing' element on the screen */
{
  setfillstyle(SOLID_FILL, COLOUR_BUTTON);
  bar(xloc, yloc, xloc+xsize-1, yloc+ysize-1);
}


void draw_nothing_text(int xloc, int yloc, int xsize, int ysize, char *text)
/* Routine to draw a 'nothing' element on the screen.  Draws */
/* text in the middle of it. */
{
  setfillstyle(SOLID_FILL, COLOUR_BUTTON);
  bar(xloc, yloc, xloc+xsize-1, yloc+ysize-1);

  setcolor(COLOUR_TEXT);
  settextjustify(CENTER_TEXT, CENTER_TEXT);
  outtextxy(xloc+xsize/2, yloc+ysize/2, text);
  settextjustify(LEFT_TEXT, TOP_TEXT);

}
```

```
void draw_cable(int xloc, int yloc, int xsize, int ysize, int type,
                int colour, int fillstyle)
/* Routine to draw a cable segment on the screen */
{
  draw_nothing(xloc, yloc, xsize, ysize);
  setfillstyle(fillstyle, colour);

  if ((type == HOR) ||
      (type == HORTTAP) ||
      (type == HORBTAP))
    bar(xloc, yloc+ysize/3, xloc+xsize-1, yloc+ysize*2/3);

  if ((type == VER) ||
      (type == VERRTAP) ||
      (type == VERLTAP))
    bar(xloc+xsize/3, yloc, xloc+xsize*2/3, yloc+ysize-1);

    if (type == HORTTAP)
      bar(xloc+xsize*2/5, yloc, xloc+xsize*3/5, yloc+ysize/3);

    if (type == HORBTAP)
      bar(xloc+xsize*2/5, yloc+ysize*2/3, xloc+xsize*3/5, yloc+ysize-1);

    if (type == VERLTAP)
      bar(xloc, yloc+ysize*2/5, xloc+xsize/3, yloc+ysize*3/5);

    if (type == VERRTAP)
      bar(xloc+xsize*2/3, yloc+ysize*2/5, xloc+xsize-1, yloc+ysize*3/5);

    if (type == RB)
    {
      if (fillstyle != SOLID_FILL)
        setcolor(COLOUR_TEXT);
      else
        setcolor(colour);
      setfillstyle(fillstyle, colour);
      sector(xloc+xsize-1, yloc+ysize-1, 90, 180, xsize*2/3-1, ysize*2/3);
      setcolor(COLOUR_BUTTON);
      setfillstyle(SOLID_FILL, COLOUR_BUTTON);
      sector(xloc+xsize-1, yloc+ysize-1, 90, 180, xsize/3-2, ysize/3-1);
    }
    else
    if (type == LB)
    {
      if (fillstyle != SOLID_FILL)
        setcolor(COLOUR_TEXT);
      else
        setcolor(colour);
```

```
      setfillstyle(fillstyle, colour);
      sector(xloc, yloc+ysize-1, 0, 90, xsize*2/3-1, ysize*2/3);
      setcolor(COLOUR_BUTTON);
      setfillstyle(SOLID_FILL, COLOUR_BUTTON);
      sector(xloc, yloc+ysize-1, 0, 90, xsize/3-2, ysize/3-1);
    }
    else
    if (type == RT)
    {
      if (fillstyle != SOLID_FILL)
        setcolor(COLOUR_TEXT);
      else
        setcolor(colour);
      setfillstyle(fillstyle, colour);
      sector(xloc+xsize-1, yloc, 180, 270, xsize*2/3-1, ysize*2/3);
      setcolor(COLOUR_BUTTON);
      setfillstyle(SOLID_FILL, COLOUR_BUTTON);
      sector(xloc+xsize-1, yloc, 180, 270, xsize/3-2, ysize/3-1);
    }
    else
    if (type == LT)
    {
      if (fillstyle != SOLID_FILL)
        setcolor(COLOUR_TEXT);
      else
        setcolor(colour);
      setfillstyle(fillstyle, colour);
      sector(xloc, yloc, 270, 360, xsize*2/3-1, ysize*2/3);
      setcolor(COLOUR_BUTTON);
      setfillstyle(SOLID_FILL, COLOUR_BUTTON);
      sector(xloc, yloc, 270, 360, xsize/3-2, ysize/3-1);
    }
}


void draw_site(int xloc, int yloc, int xsize, int ysize, int label,
               int state)
/* Routine to draw an Ethernet site on the screen */
{
  char *temptext = "........."; /* Enough space for site and labels */

  settextstyle(DEFAULT_FONT, HORIZ_DIR, 0);
  settextjustify(CENTER_TEXT, CENTER_TEXT);

  /* Draw button */
  setfillstyle(SOLID_FILL, COLOUR_BUTTON);
  bar(xloc+1, yloc+1, xloc+xsize-2, yloc+ysize-2);
```

```
/* Draw bottom and right sides */
setcolor(COLOUR_SHADOW);
line(xloc, yloc+ysize-1, xloc+xsize-1, yloc+ysize-1);
line(xloc+xsize-1, yloc, xloc+xsize-1, yloc+ysize-1);

/* Draw top and left sides */
setcolor(COLOUR_SUN);
line(xloc, yloc, xloc+xsize-1, yloc);
line(xloc, yloc, xloc, yloc+ysize-1);

/* Draw highlighting */
setcolor(COLOUR_TEXT);
if ((state != STATE_WAIT) &&
    (state != STATE_ICON) &&
    (state != STATE_MONITOR))
{
  if (state == STATE_SEND)
  {
    setcolor(COLOUR_PACKET);
    outtextxy(xloc+xsize/2, yloc+ysize/3, "Send");
  }
  else
  if (state == STATE_BACKOFF)
  {
    setcolor(COLOUR_PACKETCOLL);
    outtextxy(xloc+xsize/2, yloc+ysize/3, "Back");
  }
  else
  if (state == STATE_JAM)
  {
    setcolor(COLOUR_PACKETJAM);
    outtextxy(xloc+xsize/2, yloc+ysize/3, "JAM");
  }
  else
  if (state == STATE_SENSE)
  {
    setcolor(COLOUR_PACKETENC);
    outtextxy(xloc+xsize/2, yloc+ysize/3, "Sense");
  }
  else
  if (state == STATE_RECEIVE)
  {
    setcolor(COLOUR_PACKETENC);
    outtextxy(xloc+xsize/2, yloc+ysize/3, "Rec");
  }
}
else
  outtextxy(xloc+xsize/2, yloc+ysize/3, "Site");
```

```
  if (state != STATE_ICON)
  {
    sprintf(temptext, "%d", label);
    outtextxy(xloc+xsize/2, yloc+ysize*2/3, temptext);
  }
}


void draw_bridge(int xloc, int yloc, int xsize, int ysize,
                 int label, int state)
/* Routine to draw an Ethernet site on the screen */
{
  char *temptext = "........."; /* Enough space for site and labels */

  /* Draw button */
  setfillstyle(SOLID_FILL, COLOUR_BUTTON);
  bar(xloc+1, yloc+1, xloc+xsize-2, yloc+ysize-2);

  /* Draw bottom and right sides */
  setcolor(COLOUR_SHADOW);
  line(xloc, yloc+ysize-1, xloc+xsize-1, yloc+ysize-1);
  line(xloc+xsize-1, yloc, xloc+xsize-1, yloc+ysize-1);

  /* Draw top and left sides */
  setcolor(COLOUR_SUN);
  line(xloc, yloc, xloc+xsize-1, yloc);
  line(xloc, yloc, xloc, yloc+ysize-1);

  /* Draw bridge outline highlight */
  setcolor(COLOUR_PACKET);
  line(xloc+2, yloc+ysize-3, xloc+xsize-3, yloc+ysize-3);
  line(xloc+xsize-3, yloc+2, xloc+xsize-3, yloc+ysize-3);
  line(xloc+2, yloc+2, xloc+xsize-3, yloc+2);
  line(xloc+2, yloc+2, xloc+2, yloc+ysize-3);

  /* Draw highlighting */
  setcolor(COLOUR_TEXT);
  if ((state != STATE_WAIT) && (state != STATE_ICON))
    setcolor(COLOUR_PACKET);

  /* Draw text */
  settextstyle(DEFAULT_FONT, HORIZ_DIR, 0);
  settextjustify(CENTER_TEXT, CENTER_TEXT);

  outtextxy(xloc+xsize/2, yloc+ysize/3, "Br");

  if (state != STATE_ICON)
```

```
  {
    sprintf(temptext, "%d", label);
    outtextxy(xloc+xsize/2, yloc+ysize*2/3, temptext);
  }
}


void window_work(int outerleft, int outertop, int xsize, int ysize,
                 char *text)
/* Draw a simple work window.  These are purely decorative. */
{
  int bordersize = getmaxy()/80;
  int fontsize = getmaxy()/27;

  int outerright = outerleft+xsize-1;
  int outerbottom = outertop+ysize-1;
  int innerleft = outerleft+bordersize-1;
  int innerright = outerright-bordersize+1;
  int innertop = outertop+bordersize-1;
  int innerbottom = innertop+fontsize+2;

  setcolor(COLOUR_SHADOW);
  line(outerleft, outerbottom, outerright, outerbottom);
  line(outerright, outerbottom, outerright, outertop);
  setcolor(COLOUR_SUN);
  line(outerleft, outerbottom, outerleft, outertop);
  line(outerleft, outertop, outerright, outertop);
  setcolor(COLOUR_SHADOW);
  line(innerleft, innerbottom, innerright, innerbottom);
  line(innerright, innerbottom, innerright, innertop);
  setcolor(COLOUR_SUN);
  line(innerleft, innerbottom, innerleft, innertop);
  line(innerleft, innertop, innerright, innertop);
  setfillstyle(SOLID_FILL, COLOUR_DESKTOP);
  bar(outerleft+1, outertop+1, outerright-1, innertop-1); /* Top */
  bar(innerright+1, innertop, outerright-1, innerbottom); /* Right */
  bar(outerleft+1, innertop, innerleft-1, innerbottom); /* Left */
  bar(outerleft+1, innerbottom+1, outerright-1, outerbottom-1); /* Bottom */
  bar(innerleft+1, innertop+1, innerright-1, innerbottom-1); /* Title */
  setcolor(COLOUR_TEXT);
  settextstyle(TRIPLEX_FONT, HORIZ_DIR, 0);
  setusercharsize(1, 2, 4, 9);
  settextjustify(LEFT_TEXT, CENTER_TEXT);
  outtextxy(innerleft+bordersize, (innertop+innerbottom)/2-1, text);
}


struct icon *do_icons(struct icon *icons, int iconleft, int nettop)
```

```
/* Set up all the buttons, and gadgets on the screen */
{
  extern clock timer;

  int iconwidth = getmaxx()/14;
  int iconheight = getmaxy()/20;

  int icontop = nettop;
  int xoffset = getmaxx()/60;
  int yoffset = getmaxy()/14;
  int xspacing = getmaxx()/64;
  int yspacing = getmaxy()/48;

  int timeleft = iconleft, timetop = getmaxy()*7/10+yspacing;
  int icwidth = iconwidth, icheight = getmaxy()/20;
  int clockheight = iconheight, clockwidth = iconwidth;

  /* The Construction window */

  window_work(iconleft, nettop, iconwidth*2+xoffset*2+xspacing,
              iconheight*13, "Construction");

  /* Horizontal button */
  icons = icon_add(icons, iconleft+xoffset, icontop+yoffset,
                   iconleft+xoffset+iconwidth-1,
                   icontop+yoffset+iconheight-1, HOR, BUTTON, 0, 0);
  draw_cable(iconleft+xoffset, icontop+yoffset, iconwidth,
             iconheight, HOR, BLACK, SOLID_FILL);
  /* Horizontal with tap at top */
  icons = icon_add(icons, iconleft+xoffset,
                   icontop+yoffset+iconheight+yspacing,
                   iconleft+xoffset+iconwidth-1,
                   icontop+yoffset+iconheight+yspacing+iconheight-1,
                   HORTTAP, BUTTON, 0, 0);
  draw_cable(iconleft+xoffset, icontop+yoffset+iconheight+yspacing,
             iconwidth, iconheight, HORTTAP, BLACK, SOLID_FILL);
  /* Horizontal with tap at bottom */
  icons = icon_add(icons, iconleft+xoffset,
                   icontop+yoffset+2*iconheight+2*yspacing,
                   iconleft+xoffset+iconwidth-1,
                   icontop+yoffset+2*iconheight+2*yspacing+iconheight-1,
                   HORBTAP, BUTTON, 0, 0);
  draw_cable(iconleft+xoffset, icontop+yoffset+2*iconheight+2*yspacing,
             iconwidth, iconheight, HORBTAP, BLACK, SOLID_FILL);
  /* Down and to the right */
  icons = icon_add(icons, iconleft+xoffset,
                   icontop+yoffset+3*iconheight+3*yspacing,
                   iconleft+xoffset+iconwidth-1,
```

```
                icontop+yoffset+3*iconheight+3*yspacing+iconheight-1,
                RB, BUTTON, 0, 0);
draw_cable(iconleft+xoffset, icontop+yoffset+3*iconheight+3*yspacing,
          iconwidth, iconheight, RB, BLACK, SOLID_FILL);
/* Up and to the right */
icons = icon_add(icons, iconleft+xoffset,
                icontop+yoffset+4*iconheight+4*yspacing,
                iconleft+xoffset+iconwidth-1,
                icontop+yoffset+4*iconheight+4*yspacing+iconheight-1,
                RT, BUTTON, 0, 0);
draw_cable(iconleft+xoffset, icontop+yoffset+4*iconheight+4*yspacing,
          iconwidth, iconheight, RT, BLACK, SOLID_FILL);
/* Site */
icons = icon_add(icons, iconleft+xoffset,
                icontop+yoffset+5*iconheight+5*yspacing,
                iconleft+xoffset+iconwidth-1,
                icontop+yoffset+5*iconheight+5*yspacing+iconheight-1,
                SITE, BUTTON, 0, 0);
draw_site(iconleft+xoffset, icontop+yoffset+5*iconheight+5*yspacing,
         iconwidth, iconheight, 0, STATE_ICON);
/* Load */
icons = icon_add(icons, iconleft+xoffset,
                icontop+yoffset+6*iconheight+6*yspacing,
                iconleft+xoffset+iconwidth-1,
                icontop+yoffset+6*iconheight+6*yspacing+iconheight-1,
                LOAD, MOMENT, 0, 0);
draw_nothing_text(iconleft+xoffset,
                 icontop+yoffset+6*iconheight+6*yspacing,
                 iconwidth, iconheight, "Load");
/* Quit */
icons = icon_add(icons, iconleft+xoffset,
                icontop+yoffset+7*iconheight+7*yspacing,
                iconleft+xoffset+iconwidth-1,
                icontop+yoffset+7*iconheight+7*yspacing+iconheight-1,
                QUIT, MOMENT, 0, 0);
draw_nothing_text(iconleft+xoffset,
                 icontop+yoffset+7*iconheight+7*yspacing,
                 iconwidth, iconheight, "Quit");

/* Vertical button */
icons = icon_add(icons, iconleft+xoffset+iconwidth+xspacing,
                icontop+yoffset,
                iconleft+xoffset+iconwidth+xspacing+iconwidth-1,
                icontop+yoffset+iconheight-1, VER, BUTTON, 0, 0);
draw_cable(iconleft+xoffset+iconwidth+xspacing, icontop+yoffset,
          iconwidth, iconheight, VER, BLACK, SOLID_FILL);
/* Vertical left tap button */
icons = icon_add(icons, iconleft+xoffset+iconwidth+xspacing,
```

```
                icontop+yoffset+iconheight+yspacing,
                iconleft+xoffset+iconwidth+xspacing+iconwidth-1,
                icontop+yoffset+iconheight+yspacing+iconheight-1,
                VERLTAP, BUTTON, 0, 0);
draw_cable(iconleft+xoffset+iconwidth+xspacing,
           icontop+yoffset+iconheight+yspacing, iconwidth,
           iconheight, VERLTAP, BLACK, SOLID_FILL);
/* Vertical top tap button */
icons = icon_add(icons, iconleft+xoffset+iconwidth+xspacing,
                icontop+yoffset+2*iconheight+2*yspacing,
                iconleft+xoffset+iconwidth+xspacing+iconwidth-1,
                icontop+yoffset+2*iconheight+2*yspacing+iconheight-1,
                VERRTAP, BUTTON, 0, 0);
draw_cable(iconleft+xoffset+iconwidth+xspacing,
           icontop+yoffset+2*iconheight+2*yspacing, iconwidth,
           iconheight, VERRTAP, BLACK, SOLID_FILL);
/* Left and down */
icons = icon_add(icons, iconleft+xoffset+iconwidth+xspacing,
                icontop+yoffset+3*iconheight+3*yspacing,
                iconleft+xoffset+iconwidth+xspacing+iconwidth-1,
                icontop+yoffset+3*iconheight+3*yspacing+iconheight-1,
                LB, BUTTON, 0, 0);
draw_cable(iconleft+xoffset+iconwidth+xspacing,
           icontop+yoffset+3*iconheight+3*yspacing, iconwidth,
           iconheight, LB, BLACK, SOLID_FILL);
/* Up and left */
icons = icon_add(icons, iconleft+xoffset+iconwidth+xspacing,
                icontop+yoffset+4*iconheight+4*yspacing,
                iconleft+xoffset+iconwidth+xspacing+iconwidth-1,
                icontop+yoffset+4*iconheight+4*yspacing+iconheight-1,
                LT, BUTTON, 0, 0);
draw_cable(iconleft+xoffset+iconwidth+xspacing,
           icontop+yoffset+4*iconheight+4*yspacing, iconwidth,
           iconheight, LT, BLACK, SOLID_FILL);
/* Bridge */
icons = icon_add(icons, iconleft+xoffset+iconwidth+xspacing,
                icontop+yoffset+5*iconheight+5*yspacing,
                iconleft+xoffset+iconwidth+xspacing+iconwidth-1,
                icontop+yoffset+5*iconheight+5*yspacing+iconheight-1,
                BRIDGE, BUTTON, 0, 0);
draw_bridge(iconleft+xoffset+iconwidth+xspacing,
            icontop+yoffset+5*iconheight+5*yspacing, iconwidth,
            iconheight, 0, STATE_ICON);
/* Save */
icons = icon_add(icons, iconleft+xoffset+iconwidth+xspacing,
                icontop+yoffset+6*iconheight+6*yspacing,
                iconleft+xoffset+iconwidth+xspacing+iconwidth-1,
                icontop+yoffset+6*iconheight+6*yspacing+iconheight-1,
```

```
                    SAVE, MOMENT, 0, 0);
draw_nothing_text(iconleft+xoffset+iconwidth+xspacing,
                    icontop+yoffset+6*iconheight+6*yspacing, iconwidth,
                    iconheight, "Save");
/* Erase */
icons = icon_add(icons, iconleft+xoffset+iconwidth+xspacing,
                    icontop+yoffset+7*iconheight+7*yspacing,
                    iconleft+xoffset+iconwidth+xspacing+iconwidth-1,
                    icontop+yoffset+7*iconheight+7*yspacing+iconheight-1,
                    NOTHING, BUTTON, 0, 0);
draw_nothing_text(iconleft+xoffset+iconwidth+xspacing,
                    icontop+yoffset+7*iconheight+7*yspacing,
                    iconwidth, iconheight, "Erase");

/* The Time window */

/* Set up the time window */
window_work(timeleft, timetop, iconwidth*2+xoffset*2+xspacing,
             getmaxy()-timetop-yspacing/2, "Time");

settextstyle(DEFAULT_FONT, HORIZ_DIR, 0);

timer.init(timeleft+xspacing, timetop+yoffset, clockwidth, clockheight);
timer.update();
timer.display();

/* Reduce speed */
icons = icon_add(icons,
                    timeleft+xspacing,
                    timetop+yoffset+clockheight+2*yspacing+icheight,
                    timeleft+xspacing+icwidth-1,
                    timetop+yoffset+clockheight+2*yspacing+2*icheight-1,
                    TIME_SLOWER, MOMENT, 0, 0);

draw_nothing_text(timeleft+xspacing,
                    timetop+yoffset+clockheight+2*yspacing+icheight,
                    icwidth,
                    icheight, "<<");

/* Stop */
icons = icon_add(icons,
timeleft+xspacing,
timetop+yoffset+clockheight+yspacing,
timeleft+xspacing+icwidth-1,
timetop+yoffset+clockheight+yspacing+icheight-1, TIME_STOP, MOMENT, 0, 0);

draw_nothing_text(
timeleft+xspacing,
```

```
  timetop+yoffset+clockheight+yspacing,
  icwidth,
  icheight, "Stop");

  /* Faster */
  icons = icon_add(icons,
  timeleft+2*xspacing+icwidth,
  timetop+yoffset+clockheight+2*yspacing+icheight,
  timeleft+2*xspacing+2*icwidth-1,
  timetop+yoffset+clockheight+2*yspacing+2*icheight-1, TIME_FASTER,
  MOMENT, 0, 0);

  draw_nothing_text(
  timeleft+2*xspacing+icwidth,
  timetop+yoffset+clockheight+2*yspacing+icheight,
  icwidth,
  icheight, ">>");

  /* Step */
  icons = icon_add(icons,
  timeleft+2*xspacing+icwidth,
  timetop+yoffset+clockheight+yspacing,
  timeleft+2*xspacing+2*icwidth-1,
  timetop+yoffset+clockheight+yspacing+icheight-1, TIME_STEP, MOMENT, 0, 0);

  draw_nothing_text(
  timeleft+2*xspacing+icwidth,
  timetop+yoffset+clockheight+yspacing,
  icwidth,
  icheight, "Step");

  return icons;
}


void toggle_net_icon(struct icon *over)
{
  setcolor(COLOUR_DESKTOP);
  setlinestyle(DOTTED_LINE, 0, NORM_WIDTH);
  setwritemode(XOR_PUT);
  moveto(over->left, over->top);

  gmouse.Mshow(0);
  lineto(over->left, over->bottom);
  lineto(over->right, over->bottom);
  lineto(over->right, over->top);
  lineto(over->left, over->top);
  gmouse.Mshow(1);
```

```
  setwritemode(COPY_PUT); /* Back to normal */
  setlinestyle(SOLID_LINE, 0, NORM_WIDTH);
}


int request_unary(char *statement)
{
  int xs = getmaxx()/3, ys = getmaxy()/4;
  int l = (getmaxx()-xs)/2, t = (getmaxy()-ys)/2;
  int buttontop, buttonwidth, buttonheight;
  Mstatus position;
  int leftl, leftr, leftt, leftb;
  int state = 0; /* Nothing, or button being contacted */
  int changed = 0; /* Has the state changed since last time? */
  requestor our_requestor;

  gmouse.Mshow(0);

  if (our_requestor.init(l, t, xs, ys) != OK)
    return ERROR;

  buttontop = t+(ys-our_requestor.bordersize()-2)/2;
  buttonwidth = (xs-our_requestor.bordersize()-2)/3;
  buttonheight = (ys-our_requestor.bordersize()-2)/4;

  settextstyle(DEFAULT_FONT, HORIZ_DIR, 0); /* Print the statement */
  settextjustify(CENTER_TEXT, TOP_TEXT);
  setcolor(COLOUR_TEXT);
  outtextxy(l+(xs-our_requestor.bordersize()-2)/2,
            t+our_requestor.bordersize()*3, statement);

  /* Draw left button */
  leftl = l+buttonwidth;
  leftr = l+2*buttonwidth;
  leftt = buttontop;
  leftb = buttontop+buttonheight;

  moveto(leftl, leftb);
  setcolor(COLOUR_SHADOW);
  lineto(leftr, leftb);
  lineto(leftr, leftt);
  setcolor(COLOUR_SUN);
  lineto(leftl, leftt);
  lineto(leftl, leftb);
  settextjustify(CENTER_TEXT, CENTER_TEXT);
  setcolor(COLOUR_TEXT);
  outtextxy((leftr+leftl)/2, (leftt+leftb)/2, "OK");
```

```
/* Make mouse active, and wait for selection */

gmouse.Mshow(1);

while(1) /* First wait for user to release button! */
{
  position = gmouse.Mpos();
  if (!position.button_status)
    break;
}

while(1)
{
  position = gmouse.Mpos();
  if ((position.xaxis >= leftl) &&
      (position.xaxis <= leftr) &&
      (position.yaxis >= leftt) &&
      (position.yaxis <= leftb) &&
      (state != 1))
  {
    state = 1;
    changed = 1;
  }
  else /* We are outside the buttons */
  if (((position.xaxis < leftl) ||
       (position.xaxis > leftr) ||
       (position.yaxis < leftt) ||
       (position.yaxis > leftb)) &&
      (state != 0))
  {
    state = 0;
    changed = 1;
  }

  if (changed)
  {
    changed = 0;
    gmouse.Mshow(0);
    if (state == 0)
    { /* Draw dark text back */
      settextjustify(CENTER_TEXT, CENTER_TEXT);
      setcolor(COLOUR_TEXT);
      outtextxy((leftr+leftl)/2, (leftt+leftb)/2, "OK");
    }
    else
    if (state == 1)
    {
```

```
        settextjustify(CENTER_TEXT, CENTER_TEXT);
        setcolor(COLOUR_PACKETJAM);
        outtextxy((leftr+leftl)/2, (leftt+leftb)/2, "OK");
      }
      gmouse.Mshow(1);
    }

    if ((position.button_status) && (state != 0))
      break;
  }

  /* Turn off mouse pointer, and restore the background */

  gmouse.Mshow(0);

  our_requestor.close();

  /* Clear any outstanding button presses */

  position = gmouse.Mpressed(ButtonL);
  position = gmouse.Mpressed(ButtonR);

  return OK;
}


int request_bool(char *question)
{
  int xs = getmaxx()/3, ys = getmaxy()/4;
  int l = (getmaxx()-xs)/2, t = (getmaxy()-ys)/2;
  int buttontop, buttonwidth, buttonheight;
  Mstatus position;
  int leftl, leftr, leftt, leftb;
  int rightl, rightr, rightt, rightb;
  int answer;
  int state = 0; /* Nothing, left, or right being contacted */
  int changed = 0; /* Has the state changed since last time? */

  requestor our_requestor;

  gmouse.Mshow(0);

  if (our_requestor.init(l, t, xs, ys) != OK)
    return ERROR;

  buttontop = t+(ys-our_requestor.bordersize()-2)/2;
  buttonwidth = (xs-our_requestor.bordersize()-2)/5;
  buttonheight = (ys-our_requestor.bordersize()-2)/4;
```

```
settextstyle(DEFAULT_FONT, HORIZ_DIR, 0); /* Print the question */
settextjustify(CENTER_TEXT, TOP_TEXT);
setcolor(COLOUR_TEXT);
outtextxy(l+(xs-our_requestor.bordersize()-2)/2,
          t+our_requestor.bordersize()*3, question);

/* Draw left button */
leftl = l+buttonwidth;
leftr = l+2*buttonwidth;
leftt = buttontop;
leftb = buttontop+buttonheight;

moveto(leftl, leftb);
setcolor(COLOUR_SHADOW);
lineto(leftr, leftb);
lineto(leftr, leftt);
setcolor(COLOUR_SUN);
lineto(leftl, leftt);
lineto(leftl, leftb);
settextjustify(CENTER_TEXT, CENTER_TEXT);
setcolor(COLOUR_TEXT);
outtextxy((leftr+leftl)/2, (leftt+leftb)/2, "Yes");

/* Draw right button */
rightl = l+buttonwidth*3;
rightr = l+buttonwidth*4;
rightt = buttontop;
rightb = buttontop+buttonheight;

moveto(rightl, rightb);
setcolor(COLOUR_SHADOW);
lineto(rightr, rightb);
lineto(rightr, rightt);
setcolor(COLOUR_SUN);
lineto(rightl, rightt);
lineto(rightl, rightb);
settextjustify(CENTER_TEXT, CENTER_TEXT);
setcolor(COLOUR_TEXT);
outtextxy((rightl+rightr)/2, (rightt+rightb)/2, "No!");

/* Make mouse active, and wait for selection */
gmouse.Mshow(1);

while(1) /* First wait for user to release button! */
{
  position = gmouse.Mpos();
  if (!position.button_status)
```

```
    break;
}

while(1)
{
  position = gmouse.Mpos();
  if ((position.xaxis >= leftl) &&
      (position.xaxis <= leftr) &&
      (position.yaxis >= leftt) &&
      (position.yaxis <= leftb) &&
      (state != 1))
  {
    state = 1;
    changed = 1;
  }
  else
  if ((position.xaxis >= rightl) &&
      (position.xaxis <= rightr) &&
      (position.yaxis >= rightt) &&
      (position.yaxis <= rightb) &&
      (state != 2))
  {
    state = 2;
    changed = 1;
  }
  else /* We are outside the buttons */
  if (((position.xaxis < leftl) ||
       (position.xaxis > leftr) ||
       (position.yaxis < leftt) ||
       (position.yaxis > leftb)) &&
      ((position.xaxis < rightl) ||
       (position.xaxis > rightr) ||
       (position.yaxis < rightt) ||
       (position.yaxis > rightb)) &&
      (state != 0))
  {
    state = 0;
    changed = 1;
  }

  if (changed)
  {
    changed = 0;
    gmouse.Mshow(0);
    if (state == 0)
    { /* Draw dark text back */
      settextjustify(CENTER_TEXT, CENTER_TEXT);
      setcolor(COLOUR_TEXT);
```

```
        outtextxy((leftr+leftl)/2, (leftt+leftb)/2, "Yes");
        outtextxy((rightl+rightr)/2, (rightt+rightb)/2, "No!");
      }
      else
      if (state == 1)
      {
        settextjustify(CENTER_TEXT, CENTER_TEXT);
        setcolor(COLOUR_PACKETJAM);
        outtextxy((leftr+leftl)/2, (leftt+leftb)/2, "Yes");
        answer = YES;
      }
      else
      if (state == 2)
      {
        settextjustify(CENTER_TEXT, CENTER_TEXT);
        setcolor(COLOUR_PACKETJAM);
        outtextxy((rightl+rightr)/2, (rightt+rightb)/2, "No!");
        answer = NO;
      }
      gmouse.Mshow(1);
    }

    if ((position.button_status) && (state != 0))
      break;
  }

  /* Turn off mouse pointer, and restore the background */

  gmouse.Mshow(0);

  our_requestor.close();

  /* Clear any outstanding button presses */

  position = gmouse.Mpressed(ButtonL);
  position = gmouse.Mpressed(ButtonR);

  return answer;
}


char *request_file()
{
  requestor req;
  struct text_list *files, *temp;
  int left = getmaxx()/3;
  int top = getmaxy()/3;
  int width = getmaxx()/3;
```

```
int height = getmaxy()/3;
int bordersize = getmaxx()/100;
int font_size = 8; /* Size of the font we are using */
int pos;
char *answer = NULL;
int sr_top, sr_bottom, sr_left, sr_right; /* String requestor locations */

if (req.init(left, top, width, height) != OK)
  return NULL;

files = file_directory();

settextjustify(LEFT_TEXT, TOP_TEXT);
setcolor(COLOUR_TEXT);

/* List all the files */

temp = files;
pos = top+bordersize;
while (temp != NULL)
{
  outtextxy(left+bordersize, pos, temp->name);
  temp = temp->next;
  pos = pos+font_size+2;
}

/* Draw string requestor */

sr_top = top+height-1-req.bordersize()-bordersize*5;
sr_bottom = top+height-1-req.bordersize()-bordersize;
sr_left = left+bordersize;
sr_right = left+width-1-req.bordersize()-bordersize;

setcolor(COLOUR_TEXT);
settextjustify(LEFT_TEXT, BOTTOM_TEXT);
outtextxy(sr_left, sr_top-bordersize, "Enter filename:");

moveto(sr_left, sr_bottom);
setcolor(COLOUR_SHADOW);
lineto(sr_right, sr_bottom);
lineto(sr_right, sr_top);
setcolor(COLOUR_SUN);
lineto(sr_left, sr_top);
lineto(sr_left, sr_bottom);

/* Read in the string */

settextjustify(LEFT_TEXT, CENTER_TEXT);
```

```
  answer = read_string(sr_left+bordersize, (sr_top+sr_bottom)/2);

  req.close();

  /* Deallocate space used by file names */

  while (files != NULL)
  {
    temp = files->next;
    if (files->name != NULL)
      free(files->name);
    free(files);
    files = temp;
  }

  return answer;
}


char *describe_site_state(int state)
{
  char *answer; /* The space where the answer is to go */

  /* Allocate memory - size is not too critical */
  if ((answer = (char *)malloc(30*sizeof(char))) == NULL)
    return NULL;

  switch(state)
  {
    case STATE_JAM:
      sprintf(answer, "State: Sending JAM");
      break;
    case STATE_MONITOR:
      sprintf(answer, "State: Monitoring data");
      break;
    case STATE_WAIT:
      sprintf(answer, "State: Waiting");
      break;
    case STATE_SENSE:
      sprintf(answer, "State: Sensing");
      break;
    case STATE_SEND:
      sprintf(answer, "State: Sending packets");
      break;
    case STATE_RECEIVE:
      sprintf(answer, "State: Receiving packets");
      break;
    case STATE_BACKOFF:
```

```
      sprintf(answer, "State: Backoff");
      break;
    default:
      sprintf(answer, "State: Unknown!");
  }

  return answer;
}


char *describe_site_action(net_site *site)
{
  int state;
  char *answer;

  if (site == NULL)
    return NULL;

  if ((answer = (char *)malloc(40*sizeof(char))) == NULL)
    return NULL;

  state = site->state;

  switch(state)
  {
    case STATE_SEND:
      sprintf(answer, "Sending to site %d",
              site->dest.y*NETWIDTH+site->dest.x);
      break;
    case STATE_RECEIVE:
      sprintf(answer, "Receiving from site %d",
              site->source.y*NETWIDTH+site->source.x);
      break;
    case STATE_BACKOFF:
      sprintf(answer, "Backoff timer = %d", site->countdown);
      break;
    default:
      free(answer);
      return NULL;
  }

  return answer;
}


void display_info(netelement net[NETHEIGHT][NETWIDTH], struct icon *theicon)
{
  textwindow info;
```

```
int xs = getmaxx()/3, ys = getmaxy()/3;
int l = theicon->left+5, t = theicon->bottom+5;
Mstatus LPos, RPos;

requestor our_req;

if (our_req.init(l, t, xs, ys) != OK)
  return;

info.init(l, t, xs, ys, "Specific Info");

if (theicon->type == NETELEMENT) /* Net element selected */
{
  netelement *theitem = &net[theicon->aux][theicon->state];
  char *junk = "....................................";

  if (theitem->equipment == EQUIP_SITE)
  {
    char *answer;

    info.display("Ethernet site");
    info.display("Transmitter:");

    answer = describe_site_state(theitem->temp.site->state);
    if (answer != NULL)
    {
      info.display(answer);
      free(answer);
    }

    answer = describe_site_action(theitem->temp.site);
    if (answer != NULL)
    {
      info.display(answer);
      free(answer);
    }

    info.display("Receiver:");

    answer = describe_site_state(theitem->current.site->state);
    if (answer != NULL)
    {
      info.display(answer);
      free(answer);
    }

    answer = describe_site_action(theitem->current.site);
    if (answer != NULL)
```

```
  {
    info.display(answer);
    free(answer);
  }
} /* End of site info */

if (theitem->equipment == EQUIP_CABLE)
{
  int i; /* Dummy variable */

  info.display("Ethernet cable");

  if (theitem->current.cable->leftdir != NULL)
  {
    info.display("Left travelling data:");
    info.display("Source Dest Encrpt Type");
    for (i = 0; i < list_size(theitem->current.cable->leftdir); i++)
      info.display(describe_data(list_return(
      theitem->current.cable->leftdir, i), junk));
  } /* End of left travelling data */

  if (theitem->current.cable->rightdir != NULL)
  {
    info.display("Right travelling data:");
    info.display("Source Dest Encrpt Type");
    for (i = 0; i < list_size(theitem->current.cable->rightdir); i++)
      info.display(describe_data(list_return(
      theitem->current.cable->rightdir, i), junk));
  } /* End of right travelling data */

  if (theitem->current.cable->updir != NULL)
  {
    info.display("Upward travelling data:");
    info.display("Source Dest Encrpt Type");
    for (i = 0; i < list_size(theitem->current.cable->updir); i++)
      info.display(describe_data(list_return(
      theitem->current.cable->updir, i), junk));
  } /* End of upward travelling data */

  if (theitem->current.cable->downdir != NULL)
  {
    info.display("Downward travelling data:");
    info.display("Source Dest Encrpt Type");
    for (i = 0; i < list_size(theitem->current.cable->downdir); i++)
      info.display(describe_data(list_return(
      theitem->current.cable->downdir, i), junk));
  } /* End of downward travelling data */
```

```
} /* End of cable info */

if (theitem->equipment == EQUIP_BRIDGE)
{
  net_bridge *the_bridge = theitem->current.bridge;
  table *current = the_bridge->bridge_table;

  info.display("Ethernet bridge");

  if (current != NULL)
    info.display ("Table:");

  /* Display bridge tables */
  while (current != NULL)
  {
    sprintf(junk, "Site %d ", current->site);
    switch(current->direction)
    {
      case UP:
        strcat(junk, "is UP");
        break;
      case DOWN:
        strcat(junk, "is DOWN");
        break;
      case LEFT:
        strcat(junk, "is LEFT");
        break;
      case RIGHT:
        strcat(junk, "is RIGHT");
        break;
      default:
        strcat(junk, "???");
    }
    info.display(junk);
    current = current->next;
  }

  /* Now display buffer stats */

  if (the_bridge->outqueue.up != NULL)
  {
    bridge_buf *qp = the_bridge->outqueue.up;

    info.display("Up queue:");

    while (qp != NULL)
    {
      sprintf(junk, "From %d, to %d",
```

```
            qp->source.y*NETWIDTH+qp->source.x,
            qp->dest.y*NETWIDTH+qp->dest.x);
    info.display(junk);
    qp = qp->next;
  }
}

if (the_bridge->outqueue.down != NULL)
{
  bridge_buf *qp = the_bridge->outqueue.down;

  info.display("Down queue:");

  while (qp != NULL)
  {
    sprintf(junk, "From %d, to %d",
            qp->source.y*NETWIDTH+qp->source.x,
            qp->dest.y*NETWIDTH+qp->dest.x);
    info.display(junk);
    qp = qp->next;
  }
}

if (the_bridge->outqueue.left != NULL)
{
  bridge_buf *qp = the_bridge->outqueue.left;

  info.display("Left queue:");

  while (qp != NULL)
  {
    sprintf(junk, "From %d, to %d",
            qp->source.y*NETWIDTH+qp->source.x,
            qp->dest.y*NETWIDTH+qp->dest.x);
    info.display(junk);
    qp = qp->next;
  }
}

if (the_bridge->outqueue.right != NULL)
{
  bridge_buf *qp = the_bridge->outqueue.right;

  info.display("Right queue:");

  while (qp != NULL)
  {
    sprintf(junk, "From %d, to %d",
```

```
            qp->source.y*NETWIDTH+qp->source.x,
            qp->dest.y*NETWIDTH+qp->dest.x);
    info.display(junk);
    qp = qp->next;
  }
}

/* Describe states of sites in the bridge */

if ((the_bridge->up_t->state != STATE_WAIT) ||
    (the_bridge->down_t->state != STATE_WAIT) ||
    (the_bridge->left_t->state != STATE_WAIT) ||
    (the_bridge->right_t->state != STATE_WAIT))
{
  info.display("Active transmitters:");
  sprintf(junk, ""); /* Erase junk */

  if (the_bridge->up_t->state != STATE_WAIT)
    strcat(junk, "up ");

  if (the_bridge->down_t->state != STATE_WAIT)
    strcat(junk, "down ");

  if (the_bridge->left_t->state != STATE_WAIT)
    strcat(junk, "left ");

  if (the_bridge->right_t->state != STATE_WAIT)
    strcat(junk, "right ");

  info.display(junk);
}

if ((the_bridge->up_r->state != STATE_WAIT) ||
    (the_bridge->down_r->state != STATE_WAIT) ||
    (the_bridge->left_r->state != STATE_WAIT) ||
    (the_bridge->right_r->state != STATE_WAIT))
{
  info.display("Active receivers:");
  sprintf(junk, ""); /* Erase junk */

  if (the_bridge->up_r->state != STATE_WAIT)
    strcat(junk, "up ");

  if (the_bridge->down_r->state != STATE_WAIT)
    strcat(junk, "down ");

  if (the_bridge->left_r->state != STATE_WAIT)
    strcat(junk, "left ");
```

```
        if (the_bridge->right_r->state != STATE_WAIT)
          strcat(junk, "right ");

        info.display(junk);
      }
    } /* End of equipment bridge */
  }

  /* Make mouse active, and wait for a button click */
  gmouse.Mshow(1);
  do
  {
    LPos = gmouse.Mpressed(ButtonL);
    RPos = gmouse.Mpressed(ButtonR);

    if (kbhit() && (getch() == 59)) /* F1 key pressed */
      if (snapshot_screen() != OK)
        (void)request_unary("Snapshot failed!");
  }
  while ((LPos.button_count == 0) &&
         (RPos.button_count == 0)) ;

  /* Turn off mouse pointer, and restore the background */

  gmouse.Mshow(0);

  our_req.close();
}
```

D.5 file.h

```
#include <io.h>
#include <fcntl.h>
#include <sys\stat.h>

#include <dir.h>

struct text_list
{
  char *name;
  struct text_list *next;
```

```
};
/* The structure in which directory listings are produced.
Essentially a linked list of names. */

int save_network(netelement[NETHEIGHT][NETWIDTH], char *filename);
/* Saves the passed network into a file with the passed file name */

int load_network(netelement net[NETHEIGHT][NETWIDTH], char *filename);
/* Loads the network, in the file with the passed file name, into
the passed network */

struct text_list *file_directory(void);
/* Produces a list of files in the current directory with the
extension ".net" */
```

D.6 file.c

```
/* The file format has an end of line marker so that it's possible */
/* to read in files from networks of different dimensions. */
/* Currently this feature is not implemented, but could be */
/* if the network dimensions had to be altered for some */
/* reason. */

#define END_OF_LINE "*"

/* What follows are the characters used to represent the network */
/* in the file */

#define FILE_NOTHING "."
#define FILE_SITE "S"
#define FILE_BRIDGE "B"
#define FILE_CABLE_VER "|"
#define FILE_CABLE_VERLTAP "<"
#define FILE_CABLE_VERRTAP ">"
#define FILE_CABLE_HOR "-"
#define FILE_CABLE_HORTTAP "^"
#define FILE_CABLE_HORBTAP "v"
#define FILE_CABLE_LT "4"
#define FILE_CABLE_LB "2"
#define FILE_CABLE_RT "3"
#define FILE_CABLE_RB "1"
```

```
int save_network(netelement net[NETHEIGHT][NETWIDTH], char *filename)
{
  int i, j; /* Loop variables */
  char *item; /* The item we are writing out */
  int fh;

  if ((item = (char *)malloc(sizeof(char))) == NULL)
    return ERROR;

  /* Open file for write */

  if ((fh = creat(filename, S_IWRITE)) == -1)
    return ERROR;

  /* Write network out to file */

  for (j = 0; j < NETHEIGHT; j++)
  {
    for (i = 0; i < NETWIDTH; i++)
    {
      if (net[j][i].equipment == EQUIP_NONE)
        strncpy(item, FILE_NOTHING, 1);
      else
      if (net[j][i].equipment == EQUIP_SITE)
        strncpy(item, FILE_SITE, 1);
      else
      if (net[j][i].equipment == EQUIP_CABLE)
      {
        cable_segment *c = net[j][i].current.cable;

        switch(c->type)
        {
          case VER:
            strncpy(item, FILE_CABLE_VER, 1);
            break;
          case HOR:
            strncpy(item, FILE_CABLE_HOR, 1);
            break;
          case HORTTAP:
            strncpy(item, FILE_CABLE_HORTTAP, 1);
            break;
          case HORBTAP:
            strncpy(item, FILE_CABLE_HORBTAP, 1);
            break;
          case VERLTAP:
            strncpy(item, FILE_CABLE_VERLTAP, 1);
            break;
          case VERRTAP:
```

```
            strncpy(item, FILE_CABLE_VERRTAP, 1);
            break;
          case LT:
            strncpy(item, FILE_CABLE_LT, 1);
            break;
          case LB:
            strncpy(item, FILE_CABLE_LB, 1);
            break;
          case RT:
            strncpy(item, FILE_CABLE_RT, 1);
            break;
          case RB:
            strncpy(item, FILE_CABLE_RB, 1);
            break;
          default:
            printf("Cannot describe cable piece!\n");
        }
      }
      else
      if (net[j][i].equipment == EQUIP_BRIDGE)
        strncpy(item, FILE_BRIDGE, 1);
      else
        printf("Unknown net element!\n");

      _write(fh, item, 1);
    }
    _write(fh, &END_OF_LINE, 1);
  }

  /* Close file */

  close(fh);

  free(item);

  return OK;
}


int load_network(netelement net[NETHEIGHT][NETWIDTH], char *filename)
{
  int i, j; /* Loop variables */
  char *item; /* The item we are reading in */
  int fh;

  if ((item = (char *)malloc(sizeof(char))) == NULL)
    return ERROR;
```

```
/* Open file for read */

if ((fh = open(filename, O_RDONLY | O_BINARY)) == -1)
  return ERROR;

for (j = 0; j < NETHEIGHT; j++)
  for (i = 0; i < NETWIDTH; i++)
  {
    if (_read(fh, item, 1) != -1) /* Read char successfully */
    {
      if (!strncmp(END_OF_LINE, item, 1)) /* Skip EOLN char */
      {
        if (_read(fh, item, 1) == -1) /* Unsuccessful */
        {
          close(fh);
          return ERROR;
        }
      }

      if (!strncmp(FILE_NOTHING, item, 1))
        net[j][i].equipment = EQUIP_NONE; /* Default piece */
      else
      if (!strncmp(FILE_SITE, item, 1)) /* Create a site */
      {
        net_site *tempsite1, *tempsite2;

        tempsite1 = create_site(i, j, NULL);
        tempsite2 = create_site(i, j, NULL);

        if ((tempsite1 != NULL) && (tempsite2 != NULL))
        { /* Everything went ok */
          net[j][i].current.site = tempsite1;
          net[j][i].temp.site = tempsite2;
          net[j][i].equipment = EQUIP_SITE;
        }
        else
        { /* Memory error - free up what we did allocate */
          if (tempsite1 != NULL)
            free_site(tempsite1);
          if (tempsite2 != NULL)
            free_site(tempsite2);
        }
      }
      else
      if (!strncmp(FILE_BRIDGE, item, 1)) /* Create a bridge */
      {
        net_bridge *tempbridge;
```

```
    tempbridge = create_bridge(i, j);

    if (tempbridge != NULL)
    { /* Allocation ok */
      net[j][i].current.bridge = tempbridge;
      net[j][i].temp.bridge = NULL;
      net[j][i].equipment = EQUIP_BRIDGE;
    }
    else
      net[j][i].equipment = EQUIP_NONE;
  }
  else /* Must be cable (hopefully!) */
  {
    cable_segment *tempcable1, *tempcable2;
    int cable;

    if (!strncmp(FILE_CABLE_HOR, item, 1))
      cable = HOR;
    else
    if (!strncmp(FILE_CABLE_HORTTAP, item, 1))
      cable = HORTTAP;
    else
    if (!strncmp(FILE_CABLE_HORBTAP, item, 1))
      cable = HORBTAP;
    else
    if (!strncmp(FILE_CABLE_VER, item, 1))
      cable = VER;
    else
    if (!strncmp(FILE_CABLE_VERLTAP, item, 1))
      cable = VERLTAP;
    else
    if (!strncmp(FILE_CABLE_VERRTAP, item, 1))
      cable = VERRTAP;
    else
    if (!strncmp(FILE_CABLE_LT, item, 1))
      cable = LT;
    else
    if (!strncmp(FILE_CABLE_LB, item, 1))
      cable = LB;
    else
    if (!strncmp(FILE_CABLE_RT, item, 1))
      cable = RT;
    else
    if (!strncmp(FILE_CABLE_RB, item, 1))
      cable = RB;
    else
    {
      net[j][i].equipment = EQUIP_NONE;
```

```
            continue;
          }

          tempcable1 = create_cable(cable);
          tempcable2 = create_cable(cable);

          if ((tempcable1 != NULL) && (tempcable2 != NULL))
          { /* Allocation ok */
            net[j][i].current.cable = tempcable1;
            net[j][i].temp.cable = tempcable2;
            net[j][i].equipment = EQUIP_CABLE;
          }
          else
          { /* Allocation error */
            if (tempcable1 != NULL)
              free_cable(tempcable1);
            if (tempcable2 != NULL)
              free_cable(tempcable2);
          }
        }
      }
    }

  close(fh);

  free(item);

  return OK;
}


struct text_list *file_directory(void)
{
  struct ffblk ffblk;
  int done;
  struct text_list *files = NULL, *current;

  done = findfirst("*.net", &ffblk, 0);
  while (!done)
  {
    if ((current = (struct text_list *)malloc(sizeof(struct text_list))) ==
        NULL)
      return NULL;

    current->name = s_dup(ffblk.ff_name);
    current->next = files;
    files = current;
```

```
    done = findnext(&ffblk);
  }

  return files;
}
```

D.7 icon.h

```
struct icon
{
  struct icon *next;
  int left, top, right, bottom;
  int id;
  int type;
  int state;
  int aux;
};
/* Structure for keeping track of icons on the screen.  Essentially
a linked list of icon data. */

struct icon *icon_find(struct icon *icons, int x, int y);
/* Returns a pointer to an icon that is at the passed screen
location. */

struct icon *icon_add(struct icon *icons, int left, int top, int right,
                      int bottom, int id, int type, int state, int aux);
/* Add a new icon to the list of current icons */

void icon_highlight(struct icon *theicon);
/* Highlights an icon for easier viewing */

void icon_outline(struct icon *theicon);
/* Produces an outline around an icon, for easier viewing */

void icon_outline_all(struct icon *icons);
/* Outlines every one of a list of icons */
```

```
struct icon *find_net_icon(struct icon *icons, int i, int j);
/* Finds the icon belonging to a particular network element */
```

D.8 icon.c

```
struct icon *icon_find(struct icon *icons, int x, int y)
{
  while (icons != NULL)
  {
    if ((icons->left <= x) &&
        (icons->right >= x) &&
        (icons->top <= y) &&
        (icons->bottom >= y))
      return icons;
    icons = icons->next;
  }
  return NULL;
}


struct icon *icon_add(struct icon *icons, int left, int top, int right,
                      int bottom, int id, int type, int state, int aux)
{
  struct icon *newicon;

  if ((newicon = (struct icon *)malloc(sizeof(struct icon))) == NULL)
    return icons;

  newicon->next = icons;
  newicon->left = left;
  newicon->top = top;
  newicon->right = right;
  newicon->bottom = bottom;
  newicon->id = id;
  newicon->type = type;
  newicon->state = state;
  newicon->aux = aux;

  return newicon;
}
```

```
void icon_highlight(struct icon *theicon)
/* Takes a button icon as input, and depending on whether */
/* it has been depressed, may highlight the button. */
/* If it was not pressed, the highlight is in the background colour. */
{
  int xgap = 2, xthick = 2, ygap = 2, ythick = 2;
  int inleft, inright, intop, inbottom;
  int outleft, outright, outtop, outbottom;

  if ((theicon == NULL) || (theicon->type != BUTTON))
    return; /* Can only highlight valid, button icons */

  if (theicon->state == PRESSED)
    setfillstyle(SOLID_FILL, COLOUR_HIGHLIGHT);
  else
    setfillstyle(SOLID_FILL, COLOUR_DESKTOP); /* Not pressed */

  outleft = theicon->left-1-xgap-xthick;
  inleft = outleft+xgap;
  outright = theicon->right+1+xgap+xthick;
  inright = outright-xgap;
  outtop = theicon->top-1-ygap-ythick;
  intop = outtop+ygap;
  outbottom = theicon->bottom+1+ygap+ythick;
  inbottom = outbottom-ygap;

  bar(outleft+1, outtop, outright-1, intop); /* Top bar */
  bar(outleft+1, inbottom, outright-1, outbottom); /* Bottom bar */
  bar(outleft, outtop+1, inleft, outbottom-1); /* Left bar */
  bar(inright, outtop+1, outright, outbottom-1); /* Right bar */
}


void icon_outline(struct icon *theicon)
/* Takes a button/momentary icon as input, and produces an outline */
/* depending on whether the button is pressed or not */
{
  int col; /* The highlighting colour to use */

  if ((theicon == NULL) ||
      ((theicon->type != BUTTON) && (theicon->type != MOMENT)))
    return; /* Can only outline valid, button icons */

  if (theicon->state == PRESSED)
    setcolor(COLOUR_SUN);
  else
    setcolor(COLOUR_SHADOW); /* Not pressed */
```

```
  /* Bottom and right */
  line(theicon->left-1, theicon->bottom+1, theicon->right+1,
       theicon->bottom+1);
  line(theicon->left-2, theicon->bottom+2, theicon->right+2,
       theicon->bottom+2);
  line(theicon->right+1, theicon->bottom+1, theicon->right+1,
       theicon->top-1);
  line(theicon->right+2, theicon->bottom+2, theicon->right+2,
       theicon->top-2);

  if (theicon->state == PRESSED)
    setcolor(COLOUR_SHADOW);
  else
    setcolor(COLOUR_SUN);

  /* Left and top */
  line(theicon->left-1, theicon->bottom+1, theicon->left-1, theicon->top-1);
  line(theicon->left-2, theicon->bottom+2, theicon->left-2, theicon->top-2);
  line(theicon->left-1, theicon->top-1, theicon->right+1, theicon->top-1);
  line(theicon->left-2, theicon->top-2, theicon->right+2, theicon->top-2);

  /* Some additional highlighting for effect */

  if ((theicon->type == BUTTON) &&
      (theicon->id != SITE) &&
      (theicon->id != NOTHING) &&
      (theicon->id != BRIDGE))
  {
    if (theicon->state == PRESSED)
      col = COLOUR_PACKETENC;
    else
      col = COLOUR_TEXT;
    draw_cable(theicon->left, theicon->top,
               theicon->right-theicon->left+1,
               theicon->bottom-theicon->top+1, theicon->id, col,
               SOLID_FILL);
  }
}


void icon_outline_all(struct icon *icons)
{
  while (icons != NULL)
  {
    icon_outline(icons);
    icons = icons->next;
  }
}
```

```
struct icon *find_net_icon(struct icon *icons, int i, int j)
{
  /* Look through icons, for one at the passed net position */
  while ((icons != NULL) &&
         ((icons->type != NETELEMENT) ||
         (icons->state != i) ||
         (icons->aux != j)))
    icons = icons->next;

  return icons;
}
```

D.9 list.h

```
typedef struct
{
  int x;
  int y;
} location;

typedef struct
{
  location source;
  location dest;
  int encrypted;
  int type;
} data_packet;
/* The structure for describing a segment of a frame */

struct list_element
{
  data_packet *value;
  struct list_element *next;
};
typedef struct list_element *listptr;
/* The structure for keeping track of a number of frame
segments.  Essentially a linked list */


listptr list_add(listptr current, data_packet *newelement);
```

```
/* Add a data packet to a list of data packets.  Returns the
new list. */

int list_size(listptr head);
/* Returns the number of data packets in the data packet list */

data_packet* list_return(listptr head, int index);
/* Returns a pointer to a particular data packet.  The list is
indexed from zero upwards. */

listptr list_kill(listptr head, int killdata);
/* Deletes the whole list and frees up the memory used by
the list.  If killdata is nonzero, the user data pointed
to by the list will also be freed up. */
```

D.10 list.c

```
listptr list_add(listptr current, data_packet *newelement)
/* Adds the passed element to the list.  Returns */
/* pointer to head of the list. */
{
  listptr newlist;

  if ((newlist = (struct list_element *)malloc(sizeof(struct list_element)))
  {
    free(newelement);
    return current; /* No memory to add to list! */
  }
  newlist->value = newelement;

  if (current == NULL)
    newlist->next = NULL;
  else
    newlist->next = (struct list_element *)current;

  return newlist;
}


int list_size(listptr head)
/* Returns the number of elements in the list */
{
  int count = 0;
```

```
  while (head != NULL)
  {
    head = (listptr)head->next;
    count++;
  }

  return count;
}


data_packet* list_return(listptr head, int index)
/* Returns a pointer to a particular element (0 upwards). */
/* Returns NULL on error. */
{
  int count = 0;

  while ((count < index) && (head != NULL))
  {
    head = (listptr)head->next;
    count++;
  }

  if (head == NULL)
    return NULL;
  else
    return head->value;
}


listptr list_kill(listptr head, int killdata)
/* Deletes, and frees up the whole list. */
/* If killdata is non zero then it frees */
/* the user items as well. */
{
  listptr temp;

  while (head != NULL)
  {
    temp = (listptr)head->next;
    if (killdata != 0)
      if (head->value != NULL)
        free(head->value);
    free(head);
    head = temp;
  }

  return NULL;
```

```
}
```

D.11 se.h

```
/* SimEther, the Ethernet simulator.                                    */
/*                                                                      */
/* Written by Martin van den Nieuwelaar,                                */
/*                                                                      */
/* Around March 1993 onwards...                                         */


#include <alloc.h>
#include <malloc.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <graphics.h>
#include <math.h>

#define NETWIDTH 10 /* Width of the network in blocks */
#define NETHEIGHT 8 /* Height */

#define PACKET_LENGTH 24 /* Length in segments (blocks), of a packet */

/* Type of equipment located at a particular point on the net */

#define EQUIP_NONE 0
#define EQUIP_SITE 1
#define EQUIP_CABLE 2
#define EQUIP_BRIDGE 3

/* Propagation directions of sites (must be able to be 'anded') */

#define UP 1
#define RIGHT 2
#define DOWN 4
#define LEFT 8

/* Cable ID types */
```

```
#define HORTTAP 1
#define HORBTAP 2
#define VERLTAP 4
#define VERRTAP 8
#define HOR 16
#define VER 32
#define LB 1024
#define LT 128
#define RB 256
#define RT 512

/* Other object IDs in general */

#define NOTHING 64 /* An empty space */
#define SITE 42
#define BRIDGE 43

/* IDs for the momntary icons */

#define TIME_FASTER 65
#define TIME_SLOWER 66
#define TIME_STOP 67
#define TIME_STEP 68
#define LOAD 44
#define SAVE 45
#define QUIT 99


/* Data type */

#define HEAD 0
#define MIDDLE 1
#define TAIL 2
#define JAM 3

/* Site states */

#define STATE_SEND 0
#define STATE_BACKOFF 1
#define STATE_JAM 2
#define STATE_WAIT 3
#define STATE_SENSE 4
#define STATE_RECEIVE 6
#define STATE_MONITOR 7
#define STATE_ICON 8

/* What we can detect by looking at a piece of cable */
```

```
#define SENSE_NOTHING 0
#define SENSE_DATA 1
#define SENSE_COLLISION 2
#define SENSE_JAM 3

/* The two parts of an ethernet site */

#define TRANSMITTER 1
#define RECEIVER 2

/* The types of icons that exist */

#define BUTTON 1
#define NETELEMENT 2
#define STACK 3
#define MOMENT 4

/* All the colours we can use */

#define COLOUR_TEXT 0
#define COLOUR_NOTHING 0
#define COLOUR_PACKET 1
#define COLOUR_PACKETENC 3
#define COLOUR_PACKETJAM 2
#define COLOUR_PACKETCOLL 4
#define COLOUR_SHADOW 9
#define COLOUR_BUTTON 11
#define COLOUR_DESKTOP 12
#define COLOUR_TEXTWINDOW 13
#define COLOUR_SUN 14
#define COLOUR_HIGHLIGHT 2

/* Data encryption */

#define ENCRYPTED 1
#define NOT_ENCRYPTED 0

/* Boolean requestor uses YES/NO */

#define NO 0
#define YES 1

#define PRESSED 1
#define NOTPRESSED 0

/* Error return codes */
```

```
#define OK 0
#define ERROR -1
```

D.12 se.c

```
/*
 *             SimEther - Ethernet simulations for fun!
 */

#ifdef __TINY__
#error Graphics do not work in the tiny model
#endif

#define TITLE "SimEther V1.01"

#include "se.h"
#include "list.h"
#include "colour.h"
#include "mouse.i"
#include "window.cpp"
#include "clock.cpp"
#include "serout.h"
#include "file.h"
#include "display.h"
#include "icon.h"
#include "request.cpp"

/* Global variables */

textwindow infowindow;
textwindow actionwindow;
clock timer;

void main()
{
  Mstatus LPosition, RPosition; /* Mouse positions */
  int i,j; /* General use variables */

  int graphdriver = DETECT;
  int graphmode;

  int iconwidth, iconheight;
  int netleft, nettop, iconleft, infotop;
```

```
int xspacing, yspacing;

struct icon *currentbutton = NULL; /* Current selected button */

Mresult* Result;

struct icon *source = NULL, *dest = NULL;
/* Source and dest for packet in sim mode */

struct icon *icons = NULL; /* Must be set to NULL to start with */

int timedelay = 10; /* Initial time rate */

int bogusclicks = 0; /* Number of silly user mouse clicks */

int timepassing = 0; /* Is time initially advancing? */

struct icon *over = NULL; /* The icon we are currently over */

char *buttonactions = NULL; /* What the mouse buttons do */

int work_titlebar;

/* Net stuff */
netelement net[NETHEIGHT][NETWIDTH];

init_net(net);

detectgraph(&graphdriver, &graphmode);
if (graphdriver < 0)
{
  printf("%s\n", grapherrormsg(graphresult()));
  exit(1);
}

initgraph(&graphdriver, &graphmode, "extfiles");

if (graphdriver < 0)
{
  printf("Slight problem initialising the display!\n");
  printf("Could not find required bgi files!\n\n");
  printf("Make sure you are running SimEther from the correct directory.\n")
  exit(1);
}

set_colours();

iconwidth = getmaxx()/13;
```

```
iconheight = getmaxy()/14;
netleft = getmaxx()/100;
nettop = getmaxy()/15;
iconleft = getmaxx()*8/10;
infotop = getmaxy()*7.1/10;
xspacing = getmaxx()/128;
yspacing = getmaxy()/96;
work_titlebar = getmaxy()/19;

/* Draw title window */
window_work(0, 0, getmaxx()+1, getmaxy()+1, TITLE);

/* Hard coded info window size/location */
infowindow.init(netleft, infotop, getmaxx()*2/5,
                getmaxy()-infotop+1-yspacing, "Info Window");

/* Hard coded info window size/location */
actionwindow.init(netleft+getmaxx()*2/5+xspacing, infotop,
                  getmaxx()/3, getmaxy()-infotop+1-yspacing,
                  "Action Window");

/* Draw icons */

for (i = 0; i < NETWIDTH; i++)
  for (j = 0; j < NETHEIGHT; j++)
    icons = icon_add(icons, i*iconwidth+netleft+xspacing,
                     j*iconheight+nettop+yspacing+work_titlebar,
                     i*iconwidth+netleft+xspacing+iconwidth-1,
                     j*iconheight+nettop+work_titlebar+yspacing
                     +iconheight-1, NULL, NETELEMENT, i, j);

icons = do_icons(icons, iconleft, nettop);
icon_outline_all(icons); /* Outline the buttons */

/* Draw network */

window_work(netleft, nettop, iconwidth*NETWIDTH+2*xspacing,
            iconheight*NETHEIGHT+2*yspacing+work_titlebar, "Network");

if (load_network(net, "default.net") != OK)
{
  draw_network(net, icons);
  request_unary("No default network");
}
else
  draw_network(net, icons);
```

```
  Result = gmouse.Mreset();
  if (!Result->present)
  {
    closegraph();
    printf("Sorry, a mouse is required to use this program\n\n");
    printf("WHAT!  I hear you say.  Yep, sorry, you're just going\n");
    printf("to have to buy one...\n\n");
    printf("Alternatively, your mouse drive may not be installed correctly.\n
    exit(1); /* No mouse available */
  }

  gmouse.Mxlimit(0, getmaxx()-2);
  gmouse.Mylimit(0, getmaxy()-2);

//  fill_mem(3000);

  while (TRUE)
  {
    unsigned long i = (unsigned long)pow((double)2, (double)timedelay);

    /* Possibly put outline back on */
    gmouse.Mshow(0);
    if ((over != NULL) && (over->type == NETELEMENT))
      toggle_net_icon(over);
    gmouse.Mshow(1);

    /* Spend time waiting for mouse actions */
    while (i > 0)
    {
      LPosition = gmouse.Mpos();
      if (over != icon_find(icons, LPosition.xaxis, LPosition.yaxis))
      { /* User has moved to a different icon */
        /* Possibly deselect old icon */
        if ((over != NULL) && (over->type == NETELEMENT))
          toggle_net_icon(over);

        over = icon_find(icons, LPosition.xaxis, LPosition.yaxis);
        /* Possibly select the new icon */
        if ((over != NULL) && (over->type == NETELEMENT))
          toggle_net_icon(over);

      } /* End of user having moved to a different icon */

    {
      char *temp;
      int mouseinfox = getmaxx()*12/16, mouseinfoy = getmaxy()*28/300-1;

      temp = describe_mouse_actions(net, over, currentbutton);
```

```
  if ((temp != NULL) && (strcmp(buttonactions, temp))) /* they differ */
  {
    settextjustify(RIGHT_TEXT, CENTER_TEXT);
    settextstyle(TRIPLEX_FONT, HORIZ_DIR, 0);
    setcolor(COLOUR_DESKTOP);
    gmouse.Mshow(0);
    if (buttonactions != NULL) /* First time will be NULL */
      outtextxy(mouseinfox, mouseinfoy, buttonactions);
    free(buttonactions);
    buttonactions = temp;
    setcolor(COLOUR_TEXT);
    outtextxy(mouseinfox, mouseinfoy, buttonactions);
    gmouse.Mshow(1);
  }
  else
    if (temp != NULL)
      free(temp);
}

  LPosition = gmouse.Mpressed(ButtonL);
  RPosition = gmouse.Mpressed(ButtonR);

  if ((LPosition.button_count > 0) ||
      (RPosition.button_count > 0))
    break;

  if (timepassing != 0)
    i--;

  if (kbhit() && (getch() == 59)) /* F1 key pressed */
    if (snapshot_screen() != OK)
      (void)request_unary("Snapshot failed!");
}

/* possibly turn icon back off */
gmouse.Mshow(0);
if ((over != NULL) && (over->type == NETELEMENT))
  toggle_net_icon(over);
gmouse.Mshow(1);

if (LPosition.button_count)
{
  struct icon *pressed = icon_find(icons, LPosition.xaxis,
                                   LPosition.yaxis);
  if (pressed != NULL)
  {
    gmouse.Mshow(0);
    if (pressed->type == BUTTON) /* If it's a button, toggle */
```

```
{
  /* First turn off current icon */
  if ((currentbutton != NULL) && (currentbutton != pressed))
  {
    currentbutton->state = NOTPRESSED;
    icon_outline(currentbutton);
  }

  if (pressed->state == NOTPRESSED)
  {
    currentbutton = pressed;
    pressed->state = PRESSED;
    icon_outline(currentbutton);
  }
  else
  {
    currentbutton = NULL;
    pressed->state = NOTPRESSED;
    icon_outline(pressed);
  }
} /* End of button pressed */

if (pressed->type == NETELEMENT) /* User has clicked on the net */
{
  if ((net[pressed->aux][pressed->state].equipment != EQUIP_NONE) &&
      (((currentbutton != NULL) &&
        (currentbutton->id != NOTHING)) ||
       (currentbutton == NULL)))
  { /* There is something here already */
    /* Give information on what is here */
    display_info(net, pressed);
  }
  else
  if (currentbutton != NULL)
  {
    if (currentbutton->id == NOTHING)
    { /* Erase this spot */
      int x = pressed->state, y = pressed->aux;

      /* Deallocate space currently being used */
      if (net[y][x].equipment == EQUIP_SITE)
      {
        free_site(net[y][x].current.site);
        free_site(net[y][x].temp.site);
      }
      else
      if (net[y][x].equipment == EQUIP_CABLE)
      {
```

```
      free_cable(net[y][x].current.cable);
      free_cable(net[y][x].temp.cable);
    }
    else
    if (net[y][x].equipment == EQUIP_BRIDGE)
    {
      free_bridge(net[y][x].current.bridge);
      free_bridge(net[y][x].temp.bridge);
    }

    net[pressed->aux][pressed->state].equipment = EQUIP_NONE;
    draw_nothing(pressed->left, pressed->top,
                 pressed->right-pressed->left+1,
                 pressed->bottom-pressed->top+1);
  }
  else
  if (currentbutton->id == SITE)
  { /* Add a site */
    net_site *tempsite1, *tempsite2;

    tempsite1 = create_site(pressed->state, pressed->aux, NULL);
    tempsite2 = create_site(pressed->state, pressed->aux, NULL);

    if ((tempsite1 != NULL) && (tempsite2 != NULL))
    { /* Everything went ok */
      net[pressed->aux][pressed->state].current.site = tempsite1;
      net[pressed->aux][pressed->state].temp.site = tempsite2;
      net[pressed->aux][pressed->state].equipment = EQUIP_SITE;
      draw_site(pressed->left, pressed->top,
                pressed->right-pressed->left+1,
                pressed->bottom-pressed->top+1,
                pressed->state+pressed->aux*NETWIDTH, STATE_WAIT);
    }
    else
    { /* Memory error - free up what we did allocate */
      if (tempsite1 != NULL)
        free_site(tempsite1);
      if (tempsite2 != NULL)
        free_site(tempsite2);
    }
  }
  else
  if (currentbutton->id == BRIDGE)
  { /* Add a bridge */
    net_bridge *tempbridge;

    tempbridge = create_bridge(pressed->state, pressed->aux);
```

```
        if (tempbridge != NULL)
        { /* Allocation ok */
          net[pressed->aux][pressed->state].current.bridge = tempbridge;
          net[pressed->aux][pressed->state].temp.bridge = NULL;
          net[pressed->aux][pressed->state].equipment = EQUIP_BRIDGE;
          draw_bridge(pressed->left, pressed->top,
                      pressed->right-pressed->left+1,
                      pressed->bottom-pressed->top+1,
                      pressed->state+pressed->aux*NETWIDTH,
                      STATE_WAIT);
        }
        else
          net[pressed->aux][pressed->state].equipment = EQUIP_NONE;
      }
      else
      { /* Add a piece of cable */
        cable_segment *tempcable1, *tempcable2;

        tempcable1 = create_cable(currentbutton->id);
        tempcable2 = create_cable(currentbutton->id);

        if ((tempcable1 != NULL) && (tempcable2 != NULL))
        { /* Allocation ok */
          net[pressed->aux][pressed->state].current.cable = tempcable1;
          net[pressed->aux][pressed->state].temp.cable = tempcable2;
          net[pressed->aux][pressed->state].equipment = EQUIP_CABLE;
          draw_cable(pressed->left, pressed->top,
                     pressed->right-pressed->left+1,
                     pressed->bottom-pressed->top+1,
                     currentbutton->id, BLACK, SOLID_FILL);
        }
        else
        { /* Allocation error */
          if (tempcable1 != NULL)
            free_cable(tempcable1);
          if (tempcable2 != NULL)
            free_cable(tempcable2);
        }
      }
    }
  } /* End of net clicked */

  if (pressed->type == MOMENT) /* User has clicked a momentary switch *
  {
    pressed->state = PRESSED;
    icon_outline(pressed);

    switch (pressed->id)
```

```
{
  case TIME_FASTER:
    if (timepassing == 1)
    {
      timedelay--;
      if (timedelay < 6)
        timedelay = 6;
    }
    else
      timepassing = 1;
    break;
  case TIME_SLOWER:
    if (timepassing == 1)
    {
      timedelay++;
      if (timedelay > 15)
        timedelay = 15;
    }
    else
      timepassing = 1;
    break;
  case TIME_STOP:
    timepassing = 0;
    break;
  case TIME_STEP:
    update_net(net, icons);

    timer.update();
    timer.display();

    break;
}

if (pressed->id == QUIT)
  if (request_bool("Really quit program?") != NO)
    break;

if (pressed->id == LOAD)
{
  if (request_bool("Load network?") == YES)
  {
    char *name;

    free_net(net);
    name = request_file();

    if (load_network(net, name) != OK)
    {
```

```
            free_net(net);
            (void)request_unary("Load failed!");
          }

          draw_network(net, icons);
          free(name);
        }
      }

      if (pressed->id == SAVE)
      {
        if (request_bool("Save this network?") == YES)
        {
          char *name = request_file();

          if (save_network(net, name) != OK)
            (void)request_unary("Save failed!");

          free(name);
        }
      }

      delay(100);
      /* Deselect the momentary button */
      pressed->state = NOTPRESSED;
      icon_outline(pressed);
    } /* End of momentary switch pressed */

    icon_outline(pressed);
    gmouse.Mshow(1);
  } /* End of an icon clicked upon */
  else /* User did not click on an icon */
  {
    if (bogusclicks++ > 10)
    {
      (void)request_bool("Having fun?");

      bogusclicks = 0;
    }
  }
  gmouse.Mshow(1);
} /* End of LMB hit */
else
if (RPosition.button_count)
{
  struct icon *pressed = icon_find(icons, RPosition.xaxis,
                                   RPosition.yaxis);
```

```
  if (pressed != NULL)
  {
    if ((pressed->type == NETELEMENT) &&
        (net[pressed->aux][pressed->state].equipment == EQUIP_SITE))
    { /* User wants to set up a transmission, or similar */
      char *temptext = ".........................";

      gmouse.Mshow(0);

      if (source == NULL)
      {
        sprintf(temptext, "Source site %d",
                pressed->state+pressed->aux*NETWIDTH);
        actionwindow.display(temptext);

        source = find_net_icon(icons, pressed->state, pressed->aux);
      }
      else
      {
        sprintf(temptext, "Destination site %d",
                pressed->state+pressed->aux*NETWIDTH);
        actionwindow.display(temptext);

        dest = find_net_icon(icons, pressed->state, pressed->aux);

        /* transmit */
        if (site_transmit(net,
                          net[source->aux][source->state].temp.site,
                          source->state, source->aux, dest->state,
                          dest->aux) != OK)
          infowindow.display("Invalid transmission request!");
        source = NULL;
        dest = NULL;
      }

      gmouse.Mshow(1);
    }
  }
  else
  {
    source = NULL; /* User clicks on something else */
  }
} /* End of RMB hit */
else
{ /* Net action phase (update the net) */
  gmouse.Mshow(0);
  update_net(net, icons);
```

```
      timer.update();
      timer.display();

      gmouse.Mshow(1);
    } /* End of net action phase */
  } /* End of main loop */

  closegraph(); /* Get back to a text screen */

  printf("SimEther - written by Martin van den Nieuwelaar\n");
  printf("\nFor more information, contact me at one of the following...\n\n")
  printf("martin@cosc.canterbury.ac.nz (until 11/93)\n");
  printf("vanz@tragula.equinox.gen.nz (fast turnaround)\n");
  printf("martin_nieuwelaar@equinox.gen.nz (not so fast turnaround)\n");

  free(buttonactions);
}


#include "icon.c"
#include "display.c"
#include "serout.c"
#include "list.c"
#include "colour.c"
#include "file.c"
```

D.13 serout.h

```
struct table
{
  int direction;
  int site;
  struct table *next;
};
typedef struct table table;
/* A structure for keeping a list of site numbers, and the direction
they are in.  Designed primarily for bridge tables, so that
bridges can keep track of what sites are where. */

struct bridge_buf
{
  location source;
```

```
  location dest;
  int encrypted;
  struct bridge_buf *next;
};
typedef struct bridge_buf bridge_buf;
/* A structure for maintaining a buffer of frames to send from a site */

typedef struct
{
  int type;
  listptr leftdir;
  listptr rightdir;
  listptr updir;
  listptr downdir;
} cable_segment;
/* A structure for describing a piece of cable.  Cables can
contain data.  In this case, pointers to lists of frame
segments that propagate along the wire. */

typedef struct
{
  int connections;
  int state;
  int countdown;
  data_packet *input;
  location dest;
  location source;
  int encrypted;
  int retry;
  int automatic;
  int xloc;
  int yloc;
  int last;
} net_site;
/* A structure for describing a network site.  Describes either an
Ethernet transmitter, or a receiver. */

typedef struct
{
  bridge_buf *up, *down, *left, *right;
} bridge_buffers;
/* A structure for describing the buffers a bridge has.  In this
implementation, a bridge has a buffer for each Ethernet connection. */

typedef struct
{
  net_site *left_t, *right_t, *up_t, *down_t,
           *left_r, *right_r, *up_r, *down_r;
```

```
  table* bridge_table;
  bridge_buffers outqueue;
} net_bridge;
/* A structure that describes a bridge.  bridges are made up of
four Ethernet units, each consisting of a transmitter, and a
receiver.  Bridges also have tables to facilitate frame forwarding,
and queues for buffering outgoing frames. */

union equipment
{
  net_bridge *bridge;
  net_site* site; /* Points to an Ethernet site */
  cable_segment* cable; /* Points to a piece of cable */
};
/* A piece of equipment is either a bridge, a site, or a piece
of cable */

typedef struct
{
  int equipment;
  union equipment current;
  union equipment temp;
} netelement;
/* Each position of the network has an equipment type, as well
as two possible instances of that piece of equipment */

int snapshot_screen(void);
/* Bare bones screen snapshot routine.  Saves snapshots to the
current directory, in PPM pictre format. */

void free_site(net_site *site);
/* Frees up all memory used by a site */

void free_cable(cable_segment *cable);
/* Frees up all memory used by a piece of cable, including any data
currently on this piece of cable */

void free_bridge(net_bridge *bridge);
/* Frees up all memory used by this bridge */

void free_net(netelement net[NETHEIGHT][NETWIDTH]);
/* Frees up all the memory used by a network.  The equipment
type of each element on the network is set to empty. */

char *s_dup(char *);
/* Duplicates a string.  That is, allocates memory, and then
makes a copy. */
```

```
long mem_avail();
/* Debugging routine for testing to see how much memory is available
for allocation.  Used mainly to find memory leakages. */

bridge_buf *bridge_buf_add(bridge_buf *buffer, int srcx, int srcy,
                           int destx, int desty, int enc);
/* Add a frame to a bridge transmission buffer (FIFO).  Return the
resulting buffer. */

bridge_buf *bridge_buf_delete(bridge_buf *buffer);
/* Remove the frame at the head of the list.  Return the resulting
frame. */

int maintain_ethernet(netelement net[NETHEIGHT][NETWIDTH],
                      struct icon *icons, int xloc, int yloc);
/* Does all the housekeeping for an Ethernet device.  This involves
looking after bridges, and calling site maintenence routines. */

int maintain_site(netelement net[NETHEIGHT][NETWIDTH], net_site *site,
                  int xloc, int yloc, int mode);
/* Maintains an Ethernet site.  Housekeeping routine.  This involves
for example, sites transmitting onto a piece of cable, sensing
for collisions etc... */

table *table_delete(table *atable, int sitenum);
/* Deletes a site entry from a bridge table.  Returns the
resulting table. */

table *table_add(table *atable, int sitenum, int direction);
/* Adds site information to the bridge table.  Returns resulting
bridge table. */

int table_find(table *atable, int sitenum);
/* Searches the bridge table for a particular site.  Returns
the direction the site is connected at, should this information
be in the bridge table. */

void bridge_action(netelement net[NETHEIGHT][NETWIDTH],
                   net_bridge *this_bridge, int from);
/* Routine that does bridge housekeeping.  For example, takes
an incoming frame and updates the bridge tables.  Also takes
frames off the outgoing queue, and starts outward transmission. */

int init_net(netelement net[NETHEIGHT][NETWIDTH]);
/* Sets the entire network to contain empty spaces */

net_site *create_site(int xloc, int yloc, int direction);
/* Allocates and initialises an Ethernet site */
```

```
net_bridge *create_bridge(int x_loc, int y_loc);
/* Allocates and initialises an Ethernet bridge */

int check_site(netelement net[NETHEIGHT][NETWIDTH], int x, int y);
/* Performs a rough sanity check on a site */

cable_segment *create_cable(int type);
/* Allocates and initialises an piece of cable of the passed type */

int check_net(netelement net[NETHEIGHT][NETWIDTH]);
/* Does a rough sanity check on the passed network */

data_packet *dup_data(data_packet *source);
/* Duplicates a frame segment */

void propagate(netelement net[NETHEIGHT][NETWIDTH], int yloc, int xloc,
               listptr packetlist, int direction);
/* Propagates data on the cables in the network */

int site_connection(netelement net[NETHEIGHT][NETWIDTH],
                    int xloc, int yloc);
/* Returns the direction of connection of a site at the described
location of the passed network */

cable_segment *find_cable(netelement net[NETHEIGHT][NETWIDTH],
                          int xloc, int yloc, int connections);
/* Returns a pointer to the piece of cable the passed site is
connected to */

int check_transmission(netelement net[NETHEIGHT][NETWIDTH],
                       int xsrc, int ysrc, int xdest, int ydest);
/* Checks the validity of a transmission request.  For example
a transmission to a non-existent site is illegal. */

int transmit(netelement net[NETHEIGHT][NETWIDTH], net_site *this_site);
/* Transmits frame segments from the passed site onto the
adjoining piece of cable */

int sense_cable(cable_segment *cable);
/* Describes the state of affairs on the passed piece of cable.
For example, whether the cable is clear, or busy etc. */

void update_net(netelement net[NETHEIGHT][NETWIDTH], struct icon *icons);
/* The routine for network housekeeping.  Updates all Ethernet
devices, and propagates all signals on the network. */

int site_transmit(netelement net[NETHEIGHT][NETWIDTH],
```

```
                    net_site *this_site, int sourcex, int sourcey,
                    int destx, int desty);
/* Sets up transmission from one site to another.  Doesn't actually
do the transmission, but puts the transmitting site into the
required state. */

void fill_mem(int space);
/* Debugging routine.  Used to allocate approximately all available
memory except for the passed number of bytes.  Used to test
for example, the behaviour of routines with insufficient free
memory. */

char *describe_data(data_packet *data, char *junk);
/* Routine that produces a string containing useful information
about the passed frame segment */

char *describe_mouse_actions(netelement net[NETHEIGHT][NETWIDTH],
                             struct icon *over, struct icon *currentbutton);
/* Routine that depending on what the mouse cursor is over, and
what icons are selected, will return a string describing what
mouse button actions will have what effect */

int fill_style(cable_segment *cable);
/* Given a pointer to a piece of cable, will return a suitable style
for display on the screen.  For example, a collision will be
represented by solid fill. */

int random_num(int modnum);
/* Produces a rough pseudo-randow number in the range 0->modnum-1. */
```

D.14 serout.c

```
int snapshot_screen(void)
{
  static int count = 0;
  int fh;
  int i, j; /* Loop variables */
  int item;
  char *header = ".....................................";
  char *filename = "...............................";
  char *colour;
```

```
int red, green, blue;

sprintf(filename, "snap%d.ppm", count);

if ((colour = (char *)malloc(sizeof(char)*(getmaxx()+1)*3)) == NULL)
  return ERROR;

sound(440); /* Beep!  (Yech!) */
delay(100);
nosound();

/* Open file for write */

if ((fh = creat(filename, S_IWRITE)) == -1)
  return ERROR;

sprintf(header, "P6\n%d %d\n63\n", getmaxx()+1, getmaxy()+1);
_write(fh, header, strlen(header));

/* Dump screen */

for (j = 0; j < getmaxy()+1; j++)
{
  for (i = 0; i < getmaxx()+1; i++)
  {
    item = getpixel(i, j);
    switch(item)
    {
      case 0:
        red = 0;
        green = 0;
        blue = 0;
        break;
      case 1:
        red = 0;
        green = 35;
        blue = 0;
        break;
      case 2:
        red = 0;
        green = 12;
        blue = 50;
        break;
      case 3:
        red = 60;
        green = 30;
        blue = 0;
        break;
```

```
        case 4:
          red = 50;
          green = 0;
          blue = 0;
          break;
        default:
          red = (item-4) * 5.72;
          green = red;
          blue = red;
      }

      colour[i*3] = red;
      colour[i*3+1] = green;
      colour[i*3+2] = blue;
    }
    _write(fh, colour, (getmaxx()+1)*3);
  }

  /* Close file */

  close(fh);

  count++; /* Increment the filename count for next time */

  sound(880); /* Beep!  (Yech!) */
  delay(100);
  nosound();

  return OK;
}

void free_site(net_site *site)
{
  if (site == NULL)
    return;

  if (site->input != NULL) /* Empty the input buffer */
    free(site->input);

  free(site);
}

void free_cable(cable_segment *cable)
{
  if (cable == NULL)
    return;

  list_kill(cable->leftdir, 1); /* Delete entire list and entries */
```

```
  list_kill(cable->rightdir, 1);
  list_kill(cable->updir, 1);
  list_kill(cable->downdir, 1);

  free(cable);
}

void free_bridge(net_bridge *bridge)
{
  table *b_table, *temp_table;

  if (bridge == NULL)
    return;

  b_table = bridge->bridge_table;

  /* Free each site */

  free_site(bridge->left_r);
  free_site(bridge->right_r);
  free_site(bridge->up_r);
  free_site(bridge->down_r);

  free_site(bridge->left_t);
  free_site(bridge->right_t);
  free_site(bridge->up_t);
  free_site(bridge->down_t);

  /* Free the bridge tables */

  while (b_table != NULL)
  {
    temp_table = b_table->next;
    free(b_table);
    b_table = temp_table;
  }

  /* Free the queues to each site */

  while (bridge->outqueue.up != NULL)
    bridge->outqueue.up = bridge_buf_delete(bridge->outqueue.up);

  while (bridge->outqueue.down != NULL)
    bridge->outqueue.down = bridge_buf_delete(bridge->outqueue.down);

  while (bridge->outqueue.left != NULL)
    bridge->outqueue.left = bridge_buf_delete(bridge->outqueue.left);
```

```
  while (bridge->outqueue.right != NULL)
    bridge->outqueue.right =  bridge_buf_delete(bridge->outqueue.right);

  free(bridge);
}

void free_net(netelement net[NETHEIGHT][NETWIDTH])
{
  int i, j;

  for (j = 0; j < NETHEIGHT; j++)
    for (i = 0; i < NETWIDTH; i++)
    {
      switch (net[j][i].equipment)
      {
        case EQUIP_NONE:
          break;
        case EQUIP_SITE:
          free_site(net[j][i].current.site);
          free_site(net[j][i].temp.site);
          break;
        case EQUIP_CABLE:
          free_cable(net[j][i].current.cable);
          free_cable(net[j][i].temp.cable);
          break;
        case EQUIP_BRIDGE:
          free_bridge(net[j][i].current.bridge);
          free_bridge(net[j][i].temp.bridge);
          break;
        default:
          printf("Cannot free unknown element!\n");
      }

      net[j][i].equipment = EQUIP_NONE;
    }
}


char *s_dup(char *source)
{
  char *dest;

  if ((dest = (char *)malloc(sizeof(char)*(strlen(source)+1))) == NULL)
    return NULL;

  strcpy(dest, source);
  dest[strlen(source)] = '\0';
```

```
  return dest;
}


long mem_avail()
{
  void *memlist[1000], *temp;
  int top = -1;
  int chunk_size, max_chunk_size = 16384;
  /* maximum we can allocate at a time */
  int min_chunk_size = 8;
  long sum = 0;
  int alloc_failed;

  while ((temp = malloc(min_chunk_size)) != NULL)
  /* To the nearest few bytes */
  {
    free (temp);

    chunk_size = max_chunk_size;
    alloc_failed = 1;

    while ((chunk_size >= min_chunk_size) && (alloc_failed))
    {
      if (top > 998) /* No space left! */
      {
        printf("No room left!\n");
        exit(1);
      }

      if ((temp = malloc(chunk_size)) == NULL)
      {
        chunk_size = chunk_size/2;
        alloc_failed = 1;
      }
      else
      {
        sum += chunk_size;
        memlist[top+1] = temp;
        top++;
        alloc_failed = 0;
      }
    }
  }

  /* Deallocate it all */

  while (top > -1)
```

```
  {
    free(memlist[top]);
    top--;
  }

  return sum;
}


int random_num(int modnum)
/* Routine to produce random numbers in the range (0 - modnum-1). */
{
  static long a = 3;
  static long number = 1;

  number = (number*a)%modnum;

  return (int)number;
}


int fill_style(cable_segment *cable)
{
  listptr typical_data;

  if (cable == NULL)
    return SOLID_FILL; /* Bogus cable */

  if (cable->leftdir != NULL)
    typical_data = cable->leftdir;
  else
  if (cable->rightdir != NULL)
    typical_data = cable->rightdir;
  else
  if (cable->updir != NULL)
    typical_data = cable->updir;
  else
  if (cable->downdir != NULL)
    typical_data = cable->downdir;
  else /* There was do data on this piece of cable */
    return SOLID_FILL;

  /* Calculate the shading for this piece of cable */
  switch((list_return(typical_data, 0)->source.x+
        (list_return(typical_data, 0)->source.y)*NETWIDTH)%5)
  {
    case 0:
      return SOLID_FILL;
```

```
    case 1:
      return LINE_FILL;
    case 2:
      return SLASH_FILL;
    case 3:
      return BKSLASH_FILL;
    case 4:
      return XHATCH_FILL;
    default:
      return SOLID_FILL; /* Should never get to the default in reality */
  }
}


int init_net(netelement net[NETHEIGHT][NETWIDTH])
/* Routine to initialise the network, so that the grid is */
/* completely empty. */
{
  int i, j;

  for (i = 0; i < NETWIDTH; i++)
    for (j = 0; j < NETHEIGHT; j++)
    {
      net[j][i].equipment = EQUIP_NONE;
    }
  return OK;
}


net_site *create_site(int x_loc, int y_loc, int direction)
/* Routine to create a site, capable of sending and receiving */
/* in the given direction only */
{
  net_site *newsite;

  if ((newsite = (net_site *)malloc(sizeof(net_site))) == NULL)
    return NULL; /* Error allocating memory! */

  newsite->connections = direction;
  newsite->state = STATE_WAIT;
  newsite->countdown = 0;

  newsite->input = NULL;

  newsite->source.x = 0;
  newsite->source.y = 0;
  newsite->dest.x = 0;
  newsite->dest.y = 0;
```

```
  newsite->encrypted = 0;
  newsite->retry = 0;
  newsite->automatic = 0;
  newsite->xloc = x_loc;
  newsite->yloc = y_loc;
  newsite->last = -1;

  return newsite;
}


int table_find(table *atable, int sitenum)
{
  while ((atable != NULL) && (atable->site != sitenum))
    atable = atable->next;

  if (atable->site == sitenum)
    return atable->direction;

  return ERROR; /* Not found */
}


table *table_add(table *atable, int sitenum, int direction)
{
  table *current;

  if ((current = (table *)malloc(sizeof(table))) == NULL)
    return atable;

  current->direction = direction;
  current->site = sitenum;
  current->next = atable;

  return current;
}


table *table_delete(table *atable, int sitenum)
{
  table *current = atable;
  table *previous = NULL;

  while ((current != NULL) && (current->site != sitenum))
  {
    previous = current;
    current = current->next;
```

```
  }

  if (current == NULL)
    return atable;

  if (current == atable)
    return current->next;

  previous->next = current->next;

  return atable;
}


net_bridge *create_bridge(int x_loc, int y_loc)
/* Routine to create a bridge */
{
  net_bridge *newbridge;
  int error = 0; /* Has an error ocurred when allocing mem? */

  if ((newbridge = (net_bridge *)malloc(sizeof(net_bridge))) == NULL)
    return NULL; /* Error allocating memory! */

  if ((newbridge->left_r = create_site(x_loc, y_loc, LEFT)) == NULL)
    error = 1;

  if ((newbridge->right_r = create_site(x_loc, y_loc, RIGHT)) == NULL)
    error = 1;

  if ((newbridge->up_r = create_site(x_loc, y_loc, UP)) == NULL)
    error = 1;

  if ((newbridge->down_r = create_site(x_loc, y_loc, DOWN)) == NULL)
    error = 1;

  if ((newbridge->left_t = create_site(x_loc, y_loc, LEFT)) == NULL)
    error = 1;

  if ((newbridge->right_t = create_site(x_loc, y_loc, RIGHT)) == NULL)
    error = 1;

  if ((newbridge->up_t = create_site(x_loc, y_loc, UP)) == NULL)
    error = 1;

  if ((newbridge->down_t = create_site(x_loc, y_loc, DOWN)) == NULL)
    error = 1;

  if (error) /* Free up what sites we did alloc, and return */
```

```
  {
    if (newbridge->left_r)
      free_site(newbridge->left_r);
    if (newbridge->right_r)
      free_site(newbridge->right_r);
    if (newbridge->up_r)
      free_site(newbridge->up_r);
    if (newbridge->down_r)
      free_site(newbridge->down_r);
    if (newbridge->left_t)
      free_site(newbridge->left_t);
    if (newbridge->right_t)
      free_site(newbridge->right_t);
    if (newbridge->up_t)
      free_site(newbridge->up_t);
    if (newbridge->down_t)
      free_site(newbridge->down_t);

    free(newbridge);

    return NULL;
  }

  newbridge->outqueue.left = NULL;
  newbridge->outqueue.right = NULL;
  newbridge->outqueue.up = NULL;
  newbridge->outqueue.down = NULL;

  newbridge->bridge_table = NULL;

  /* Create sites and join them in to the bridge */

  return newbridge;
}


int check_site(netelement net[NETHEIGHT][NETWIDTH], int x, int y)
{
  if ((x >= NETWIDTH) ||
      (x < 0) ||
      (y >= NETHEIGHT) ||
      (y < 0) ||
      (net[y][x].equipment != EQUIP_SITE) ||
      (net[y][x].current.site == NULL) ||
      (net[y][x].temp.site == NULL))
    return ERROR;

  return OK;
```

```
}


cable_segment *create_cable(int type)
/* Routine to create a cable segment of the given type */
{
  cable_segment *newcable;

  if ((newcable = (cable_segment *)malloc(sizeof(cable_segment))) == NULL)
    return NULL; /* Error allocating memory! */

  newcable->type = type;

  newcable->leftdir = NULL;
  newcable->rightdir = NULL;
  newcable->updir = NULL;
  newcable->downdir = NULL;

  return newcable;
}


int check_net(netelement net[NETHEIGHT][NETWIDTH])
/* Does a basic cable and site layout check to make sure the network */
/* proposed is a valid one. */
/* Returns either OK or ERROR. */
{
  int i, j;

  for (j = 0; j < NETHEIGHT; j++)
    for (i = 0; i < NETWIDTH; i++)
    {
      if (net[j][i].equipment == EQUIP_CABLE)
      {
        if ((net[j][i].current.cable == NULL) ||
            (net[j][i].temp.cable == NULL))
        {
          printf("Cables not installed properly!\n");
          return ERROR;
        }
        if (net[j][i].current.cable->type != net[j][i].temp.cable->type)
        {
          printf("Cable type inconsistency!\n");
          return ERROR;
        }
      }
    }
```

```
  return OK;
}


data_packet *dup_data(data_packet *source)
/* Allocate space for a new data packet */
/* and copy across information from the */
/* source packet */
{
  data_packet *dest;

  if (source == NULL)
    return NULL;

  if ((dest = (data_packet *)malloc(sizeof(data_packet))) == NULL)
    return NULL;

  dest->source.x = source->source.x;
  dest->source.y = source->source.y;
  dest->dest.x = source->dest.x;
  dest->dest.y = source->dest.y;
  dest->encrypted = source->encrypted;
  dest->type = source->type;

  return dest;
}


void propagate(netelement net[NETHEIGHT][NETWIDTH], int yloc, int xloc,
               listptr packetlist, int direction)
/* Routine to move packets from one segment to another. */
/* Actually propagates from the current list, to the temp */
/* list. */
{
  int loop;
  int total = list_size(packetlist);

  for (loop = 0; loop < total; loop++)
  {
    data_packet *temppacket = dup_data(list_return(packetlist, loop));

    /* Propagate along wires */

    if ((direction == RIGHT) && (temppacket != NULL))
    {
      /* Destroy packets that reach a terminator */
      if ((xloc+1 >= NETWIDTH) ||
          (net[yloc][xloc+1].equipment != EQUIP_CABLE) ||
```

```
      ((net[yloc][xloc+1].temp.cable->type &
        (HOR | HORTTAP | HORBTAP | LB | LT)) == 0))
    free(temppacket);
  else
  if (net[yloc][xloc+1].temp.cable->type & LT)
    net[yloc][xloc+1].temp.cable->updir =
      list_add(net[yloc][xloc+1].temp.cable->updir, temppacket);
  else
  if (net[yloc][xloc+1].temp.cable->type & LB)
    net[yloc][xloc+1].temp.cable->downdir =
      list_add(net[yloc][xloc+1].temp.cable->downdir, temppacket);
  else
    net[yloc][xloc+1].temp.cable->rightdir =
      list_add(net[yloc][xloc+1].temp.cable->rightdir, temppacket);
} /* End of direction equals right */

if ((direction == LEFT) && (temppacket != NULL))
{ /* Need to make these modifications for other parts too!!! */
  if ((xloc-1 < 0) ||
      (net[yloc][xloc-1].equipment != EQUIP_CABLE) ||
      ((net[yloc][xloc-1].temp.cable->type &
        (HOR | HORTTAP | HORBTAP | RB | RT)) == 0))
    free(temppacket);
  else
  if (net[yloc][xloc-1].temp.cable->type & RB)
    net[yloc][xloc-1].temp.cable->downdir =
      list_add(net[yloc][xloc-1].temp.cable->downdir, temppacket);
  else
  if (net[yloc][xloc-1].temp.cable->type & RT)
    net[yloc][xloc-1].temp.cable->updir =
      list_add(net[yloc][xloc-1].temp.cable->updir, temppacket);
  else
    net[yloc][xloc-1].temp.cable->leftdir =
      list_add(net[yloc][xloc-1].temp.cable->leftdir, temppacket);
} /* End of direction equals left */

if ((direction == UP) && (temppacket != NULL))
{
  /* Destroy packets that reach a terminator */
  if ((yloc-1 < 0) ||
      (net[yloc-1][xloc].equipment != EQUIP_CABLE) ||
      ((net[yloc-1][xloc].temp.cable->type &
        (VER | VERLTAP | VERRTAP | RB | LB)) == 0))
    free(temppacket);
  else
    if (net[yloc-1][xloc].temp.cable->type & RB)
      net[yloc-1][xloc].temp.cable->rightdir =
        list_add(net[yloc-1][xloc].temp.cable->rightdir, temppacket);
```

```
      else
        if (net[yloc-1][xloc].temp.cable->type & LB)
          net[yloc-1][xloc].temp.cable->leftdir =
            list_add(net[yloc-1][xloc].temp.cable->leftdir, temppacket);
      else
        net[yloc-1][xloc].temp.cable->updir =
          list_add(net[yloc-1][xloc].temp.cable->updir, temppacket);
    } /* End of direction equals right */

    if ((direction == DOWN) && (temppacket != NULL))
    {
      /* Destroy packets that reach a terminator */
      if ((yloc+1 >= NETHEIGHT) ||
          (net[yloc+1][xloc].equipment != EQUIP_CABLE) ||
          ((net[yloc+1][xloc].temp.cable->type &
            (VER | VERLTAP | VERRTAP | RT | LT)) == 0))
        free(temppacket);
      else
        if (net[yloc+1][xloc].temp.cable->type & RT)
          net[yloc+1][xloc].temp.cable->rightdir =
            list_add(net[yloc+1][xloc].temp.cable->rightdir, temppacket);
      else
        if (net[yloc+1][xloc].temp.cable->type & LT)
          net[yloc+1][xloc].temp.cable->leftdir =
            list_add(net[yloc+1][xloc].temp.cable->leftdir, temppacket);
      else
        net[yloc+1][xloc].temp.cable->downdir =
          list_add(net[yloc+1][xloc].temp.cable->downdir, temppacket);
    } /* End of direction equals down */

  }
}


int site_connection(netelement net[NETHEIGHT][NETWIDTH], int xloc, int yloc)
/* Routine to find direction of connection to network from this site */
{
  if (check_site(net, xloc, yloc) != OK)
    return ERROR;

  if ((yloc+1 < NETHEIGHT) &&
      (net[yloc+1][xloc].equipment == EQUIP_CABLE) &&
      (net[yloc+1][xloc].current.cable->type == HORTTAP))
    return DOWN; /* Connected at the bottom */
  else
  if ((yloc-1 >= 0) &&
      (net[yloc-1][xloc].equipment == EQUIP_CABLE) &&
      (net[yloc-1][xloc].current.cable->type == HORBTAP))
```

```
    return UP; /* Connected at the top */
  else
  if ((xloc-1 >= 0) &&
      (net[yloc][xloc-1].equipment == EQUIP_CABLE) &&
      (net[yloc][xloc-1].current.cable->type == VERRTAP))
    return LEFT; /* Connected at the left */
  else
  if ((xloc+1 < NETWIDTH) &&
      (net[yloc][xloc+1].equipment == EQUIP_CABLE) &&
      (net[yloc][xloc+1].current.cable->type == VERLTAP))
    return RIGHT; /* Connected at the right */

  return ERROR;
}


cable_segment *find_cable(netelement net[NETHEIGHT][NETWIDTH], int xloc, int
/* Routine to find the piece of cable joined to this site */
{
  int destx = xloc, desty = yloc;

  if (direction == 0) /* Site is not connected anywhere in particular */
  {
    direction = site_connection(net, xloc, yloc);
  }

  if ((direction == DOWN) &&
      (yloc+1 < NETHEIGHT) &&
      (net[yloc+1][xloc].equipment == EQUIP_CABLE) &&
      (net[yloc+1][xloc].current.cable->type == HORTTAP))
    desty++; /* Connected at the bottom */
  else
  if ((direction == UP) &&
     (yloc-1 >= 0) &&
     (net[yloc-1][xloc].equipment == EQUIP_CABLE) &&
     (net[yloc-1][xloc].current.cable->type == HORBTAP))
    desty--; /* Connected at the top */
  else
  if ((direction == LEFT) &&
      (xloc-1 >= 0) &&
      (net[yloc][xloc-1].equipment == EQUIP_CABLE) &&
      (net[yloc][xloc-1].current.cable->type == VERRTAP))
    destx--; /* Connected at the left */
  else
  if ((direction == RIGHT) &&
      (xloc+1 < NETWIDTH) &&
      (net[yloc][xloc+1].equipment == EQUIP_CABLE) &&
      (net[yloc][xloc+1].current.cable->type == VERLTAP))
```

```
    destx++; /* Connected at the right */
  else /* Could not find a piece of cable nearby */
    return NULL;

  if (net[desty][destx].equipment != EQUIP_CABLE)
    return NULL;

  return(net[desty][destx].current.cable);
}


int check_transmission(netelement net[NETHEIGHT][NETWIDTH], int xsrc,
                       int ysrc, int xdest, int ydest)
{
  /* Check it's a valid request */
  if ((xsrc < 0) ||
      (xsrc >= NETWIDTH) ||
      (ysrc < 0) ||
      (ysrc >= NETHEIGHT) ||
      (xdest < 0) ||
      (xdest >= NETWIDTH) ||
      (ydest < 0) ||
      (ydest >= NETHEIGHT) ||
      (net[ydest][xdest].equipment != EQUIP_SITE))
    return ERROR;

  return OK;
}


int transmit(netelement net[NETHEIGHT][NETWIDTH], net_site *this_site)
{ /* Transmits onto the temporary list */
  int direction; /* Direction of transmission */
  data_packet *packet1;
  data_packet *packet2;
  int xsrc = this_site->xloc, ysrc = this_site->yloc;

  packet1 = (data_packet *)malloc(sizeof(data_packet));
  packet2 = (data_packet *)malloc(sizeof(data_packet));

  if ((packet1 == NULL) || (packet2 == NULL))
  {
    if (packet1 != NULL)
      free(packet1);
    if (packet2 != NULL)
      free(packet2);

    return ERROR;
```

```
}

packet1->source.x = this_site->source.x;
packet1->source.y = this_site->source.y;
packet1->dest.x = this_site->dest.x;
packet1->dest.y = this_site->dest.y;
packet1->encrypted = this_site->encrypted;
if (this_site->state == STATE_JAM)
{
  packet1->type = JAM;
  packet1->source.x = this_site->xloc;
  packet1->source.y = this_site->yloc;
  packet1->dest.x = -1;
  packet1->dest.y = 0;
}
else
if (this_site->countdown == PACKET_LENGTH)
  packet1->type = HEAD;
else
if (this_site->countdown == 1) /* Last chunk to send */
  packet1->type = TAIL;
else
  packet1->type = MIDDLE;

packet2->source.x = this_site->source.x;
packet2->source.y = this_site->source.y;
packet2->dest.x = this_site->dest.x;
packet2->dest.y = this_site->dest.y;
packet2->encrypted = this_site->encrypted;
if (this_site->state == STATE_JAM)
{
  packet2->type = JAM;
  packet2->source.x = this_site->xloc;
  packet2->source.y = this_site->yloc;
  packet2->dest.x = -1;
  packet2->dest.y = 0;
}
else
if (this_site->countdown == PACKET_LENGTH)
  packet2->type = HEAD;
else
if (this_site->countdown == 1) /* Last chunk to send */
  packet2->type = TAIL;
else
  packet2->type = MIDDLE;

if (this_site->connections == 0)
  direction = site_connection(net, xsrc, ysrc);
```

```
  else
    direction = this_site->connections;

  if (direction == DOWN)
  {
    net[ysrc+1][xsrc].temp.cable->leftdir =
      list_add(net[ysrc+1][xsrc].temp.cable->leftdir, packet1);
    net[ysrc+1][xsrc].temp.cable->rightdir =
      list_add(net[ysrc+1][xsrc].temp.cable->rightdir, packet2);
  }
  else
  if (direction == UP)
  {
    net[ysrc-1][xsrc].temp.cable->leftdir =
      list_add(net[ysrc-1][xsrc].temp.cable->leftdir, packet1);
    net[ysrc-1][xsrc].temp.cable->rightdir =
      list_add(net[ysrc-1][xsrc].temp.cable->rightdir, packet2);
  }
  else
  if (direction == LEFT)
  {
    net[ysrc][xsrc-1].temp.cable->updir =
      list_add(net[ysrc][xsrc-1].temp.cable->updir, packet1);
    net[ysrc][xsrc-1].temp.cable->downdir =
      list_add(net[ysrc][xsrc-1].temp.cable->downdir, packet2);
  }
  else
  if (direction == RIGHT)
  {
    net[ysrc][xsrc+1].temp.cable->updir =
      list_add(net[ysrc][xsrc+1].temp.cable->updir, packet1);
    net[ysrc][xsrc+1].temp.cable->downdir =
      list_add(net[ysrc][xsrc+1].temp.cable->downdir, packet2);
  }
  else
  {
    printf("Cannot find a suitable direction to transmit in!\n");
    return ERROR;
  }

  return OK;
}


int sense_cable(cable_segment *cable)
/* Routine to describe what data is on the passed cable */
{
  listptr current;
```

```
int datastate = SENSE_NOTHING; /* The most 'severe' data on the cable */

if (cable == NULL)
  return ERROR; /* Can't sense on bogus cable! */

/* Remember to add code for up/down directions!!! */

if ((list_size(cable->leftdir) > 0) ||
    (list_size(cable->rightdir) > 0) ||
    (list_size(cable->updir) > 0) ||
    (list_size(cable->downdir) > 0))
  datastate = SENSE_DATA;


/* Perhaps the following could be shortened somewhat... */

if ((list_size(cable->leftdir) == 1) && /* Possible jam case */
    (list_size(cable->rightdir) == 1))
  if ((cable->leftdir->value->source.x != cable->rightdir->value->source.x)
      (cable->leftdir->value->source.y != cable->rightdir->value->source.y)
    datastate = SENSE_COLLISION;

if ((list_size(cable->updir) == 1) && /* Possible jam case */
    (list_size(cable->downdir) == 1))
  if ((cable->updir->value->source.x != cable->downdir->value->source.x) ||
      (cable->updir->value->source.y != cable->downdir->value->source.y))
    datastate = SENSE_COLLISION;

if ((list_size(cable->updir) == 1) && /* Possible jam case */
    (list_size(cable->leftdir) == 1))
  if ((cable->updir->value->source.x != cable->leftdir->value->source.x) ||
      (cable->updir->value->source.y != cable->leftdir->value->source.y))
    datastate = SENSE_COLLISION;

if ((list_size(cable->updir) == 1) && /* Possible jam case */
    (list_size(cable->rightdir) == 1))
  if ((cable->updir->value->source.x != cable->rightdir->value->source.x) |
      (cable->updir->value->source.y != cable->rightdir->value->source.y))
    datastate = SENSE_COLLISION;

if ((list_size(cable->leftdir) == 1) && /* Possible jam case */
    (list_size(cable->downdir) == 1))
  if ((cable->leftdir->value->source.x != cable->downdir->value->source.x)
      (cable->leftdir->value->source.y != cable->downdir->value->source.y))
    datastate = SENSE_COLLISION;

if ((list_size(cable->rightdir) == 1) && /* Possible jam case */
    (list_size(cable->downdir) == 1))
```

```
    if ((cable->rightdir->value->source.x != cable->downdir->value->source.x) |
        (cable->rightdir->value->source.y != cable->downdir->value->source.y))
      datastate = SENSE_COLLISION;


  if ((list_size(cable->leftdir) > 1) || /* Four definite collision cases */
      (list_size(cable->rightdir) > 1) ||
      (list_size(cable->updir) > 1) ||
      (list_size(cable->downdir) > 1))
    datastate = SENSE_COLLISION;

  current = cable->leftdir;
  while (current != NULL)
  {
    if (current->value->type == JAM)
      datastate = SENSE_JAM;
    current = current->next;
  }

  current = cable->rightdir;
  while (current != NULL)
  {
    if (current->value->type == JAM)
      datastate = SENSE_JAM;
    current = current->next;
  }

  current = cable->updir;
  while (current != NULL)
  {
    if (current->value->type == JAM)
      datastate = SENSE_JAM;
    current = current->next;
  }

  current = cable->downdir;
  while (current != NULL)
  {
    if (current->value->type == JAM)
      datastate = SENSE_JAM;
    current = current->next;
  }

  return datastate;
}


int maintain_site(netelement net[NETHEIGHT][NETWIDTH], net_site *site,
```

```
                    int xloc, int yloc, int mode)
/* This is the general routine for taking care of a site */
{
  char *temptext = "................................";

  if ((mode != TRANSMITTER) && (mode != RECEIVER))
  {
    printf("Not suitable mode!\n");
    return ERROR;
  }

  if ((mode == RECEIVER) && (site->input == NULL))
  /* put data in buffer from cable */
  {
    cable_segment *the_cable = find_cable(net, xloc, yloc, site->connections)

    if ((sense_cable(the_cable) == SENSE_DATA) &&
        (list_size(the_cable->leftdir)+
          list_size(the_cable->rightdir)+
          list_size(the_cable->updir)+list_size(the_cable->downdir) == 1))
    { /* There is data there to get */
      data_packet *temp_data;

      if (list_size(the_cable->leftdir))
        temp_data = list_return(the_cable->leftdir, 0);
      else
      if (list_size(the_cable->rightdir))
        temp_data = list_return(the_cable->rightdir, 0);
      else
      if (list_size(the_cable->updir))
        temp_data = list_return(the_cable->updir, 0);
      else
      if (list_size(the_cable->downdir))
        temp_data = list_return(the_cable->downdir, 0);
      else
      {
        printf("Sanity check failure!\n");
        exit(1);
      }

      site->input = dup_data(temp_data);
    }
  } /* End of putting data in buffer from cable */

  if (site->state == STATE_SENSE)            /* Sense */
  {
    if (sense_cable(find_cable(net, xloc, yloc, site->connections)) ==
    SENSE_NOTHING)
```

```
  {
    sprintf(temptext, "%d begins transmission", xloc+yloc*NETWIDTH);
    infowindow.display(temptext);
    site->state = STATE_SEND; /* Nothing on the cable */
  }
  else
  if (find_cable(net, xloc, yloc, site->connections) == NULL)
  { /* No cable - give up */
    site->state = STATE_WAIT;
  }
} /* End sense */
else
if (site->state == STATE_SEND)              /* Send */
  if (site->countdown > 0)
  {
    if (transmit(net, site) != OK)
      return ERROR;

    site->countdown--;

    if (sense_cable(find_cable(net, xloc, yloc, site->connections)) ==
    SENSE_COLLISION)
    {
      sprintf(temptext, "%d detects collision", xloc+yloc*NETWIDTH);
      infowindow.display(temptext);
      sprintf(temptext, "   begins JAMMING", xloc+yloc*NETWIDTH);
      infowindow.display(temptext);

      site->countdown = 1; /* Collision detected */
      site->state = STATE_JAM;
    }
    else
    if (sense_cable(find_cable(net, xloc, yloc, site->connections)) ==
    SENSE_JAM)
    {
      sprintf(temptext, "%d detects JAM signal", xloc+yloc*NETWIDTH);
      infowindow.display(temptext);

      site->countdown = PACKET_LENGTH; /* JAM detected */
      site->state = STATE_BACKOFF;
    }
  }
  else
  {
    sprintf(temptext, "%d finishes transmission", xloc+yloc*NETWIDTH);
    infowindow.display(temptext);
    site->state = STATE_WAIT;
  }
```

```
else
if (site->state == STATE_JAM)                /* Jam */
{
  if (site->countdown > 0)
  {
    site->countdown--;

    if (transmit(net, site) != OK)
      return ERROR;
  }
  else
  {
    sprintf(temptext, "%d backing off", xloc+yloc*NETWIDTH);
    infowindow.display(temptext);

    site->countdown = random_num(100)/2;
    site->state = STATE_BACKOFF;
  }
} /* End jam */
else
if (site->state == STATE_WAIT)               /* Wait */
{
  if (sense_cable(find_cable(net, xloc, yloc, site->connections)) ==
  SENSE_COLLISION)
  {
    /* Do nothing if we see a collision */
  }
  else
  if ((site->input != NULL) && (site->input->type == HEAD))
  {
    if ((site->input->dest.x == site->xloc) &&
        (site->input->dest.y == site->yloc) &&
        ((site->input->source.x != site->xloc) ||
         (site->input->source.y != site->yloc)))
    { /* Data is destined for this site, and is not from it */
      sprintf(temptext, "%d begins reception", xloc+yloc*NETWIDTH);
      infowindow.display(temptext);
      site->dest.x = site->input->dest.x;
      site->dest.y = site->input->dest.y;
      site->source.x = site->input->source.x;
      site->source.y = site->input->source.y;
      site->state = STATE_RECEIVE;
    }
    else
    { /* Data is for another site */
      site->state = STATE_MONITOR;
    }
  }
```

```
  if (site->input != NULL)
  {
    /* Remove the bogus packet from the list (free memory) */
    free(site->input);
    site->input = NULL;
  }
} /* End wait */
else
if (site->state == STATE_RECEIVE)          /* Receive */
{
  if (site->input != NULL)
  {
    if (site->input->type == TAIL)
    {
      sprintf(temptext, "%d finishes reception", xloc+yloc*NETWIDTH);
      infowindow.display(temptext);
      site->last = site->input->source.y*NETWIDTH+site->input->source.x;

      site->state = STATE_WAIT;
    }

    free(site->input);
    site->input = NULL;

    if (sense_cable(find_cable(net, xloc, yloc, site->connections)) ==
    SENSE_COLLISION)
    {
      sprintf(temptext, "%d collision detected", xloc+yloc*NETWIDTH);
      infowindow.display(temptext);

      site->state = STATE_WAIT;
    }
  }
  else /* Break in the transmission! */
  {
    sprintf(temptext, "%d truncated input", xloc+yloc*NETWIDTH);
    infowindow.display(temptext);

    site->state = STATE_WAIT;
  }
} /* End receive */
else
if (site->state == STATE_MONITOR)          /* Monitor */
{
  if (site->input != NULL)
  {
    if (site->input->type == TAIL)
```

```
      {
        site->last = site->input->source.y*NETWIDTH+site->input->source.x;
        site->source.x = site->input->source.x;
        site->source.y = site->input->source.y;
        site->dest.x = site->input->dest.x;
        site->dest.y = site->input->dest.y;

        site->state = STATE_WAIT;
      }

      free(site->input);
      site->input = NULL;

      if (sense_cable(find_cable(net, xloc, yloc, site->connections)) ==
      SENSE_COLLISION)
      {
        infowindow.display("monitor coll!");
        site->state = STATE_WAIT;
      }
    }
    else /* Break in the transmission! */
    {
      site->state = STATE_WAIT;
    }
  } /* End monitor */
  else
  if (site->state == STATE_BACKOFF)          /* Backoff */
  {
    if (site->countdown > 0)
      site->countdown--;
    else
    {
      sprintf(temptext, "%d sensing for carrier", xloc+yloc*NETWIDTH);
      infowindow.display(temptext);
      site->countdown = PACKET_LENGTH;
      site->state = STATE_SENSE;
    }
  } /* End backoff */

  return OK;
}


bridge_buf *bridge_buf_add(bridge_buf *buffer, int srcx, int srcy,
                           int destx, int desty, int enc)
{
  bridge_buf *new_buf, *current = buffer, *previous = NULL;
```

```
  if ((new_buf = (bridge_buf *)malloc(sizeof(bridge_buf))) == NULL)
    return buffer;

  new_buf->source.x = srcx;
  new_buf->source.y = srcy;
  new_buf->dest.x = destx;
  new_buf->dest.y = desty;
  new_buf->encrypted = enc;
  new_buf->next = NULL;

  while (current != NULL)
  {
    previous = current;
    current = current->next;
  }

  if (previous == NULL) /* Adding to an empty list */
    return new_buf;

  previous->next = new_buf;

  return buffer;
}


bridge_buf *bridge_buf_delete(bridge_buf *buffer)
{
  if (buffer != NULL)
  {
    bridge_buf *temp = buffer->next;

    free(buffer);
    return temp;
  }

  return NULL;
}


void bridge_action(netelement net[NETHEIGHT][NETWIDTH],
                   net_bridge *this_bridge, int from)
/* This routine will possibly update the bridge tables.  It */
/* will also pass on the message to the output sites. */
{
  net_site *source;

  if (this_bridge == NULL)
  {
```

```
  printf("Invalid bridge!\n");
  exit(1);
}

if (from == LEFT)
  source = this_bridge->left_r;
else
if (from == RIGHT)
  source = this_bridge->right_r;
else
if (from == UP)
  source = this_bridge->up_r;
else
if (from == DOWN)
  source = this_bridge->down_r;
else
{
  printf("Invalid source!\n");
  exit(1);
}

if (table_find(this_bridge->bridge_table, source->last) == ERROR)
{ /* Message source is not in bridge table - add it */
  char *text = (char *)malloc(100*sizeof(char)); /* Some space for text */

  if (text == NULL)
    return;

  sprintf(text, "%d adding %d to bridge table",
          this_bridge->up_r->yloc*NETWIDTH+this_bridge->up_r->xloc,
          source->last);
  infowindow.display(text);

  this_bridge->bridge_table = table_add(this_bridge->bridge_table,
                                        source->last, from);
}

if (table_find(this_bridge->bridge_table,
               source->dest.y*NETWIDTH+source->dest.x) == ERROR)
{ /* Destination not listed - send out on all other connections */
  if ((from != UP) && (find_cable(net, this_bridge->up_r->xloc,
  this_bridge->up_r->yloc, this_bridge->up_r->connections) != NULL))
  {
    this_bridge->outqueue.up = bridge_buf_add(this_bridge->outqueue.up,
    source->source.x, source->source.y, source->dest.x, source->dest.y,
    NOT_ENCRYPTED);
  }
  if ((from != RIGHT) && (find_cable(net, this_bridge->right_r->xloc,
```

```
  this_bridge->right_r->yloc, this_bridge->right_r->connections) != NULL))
    this_bridge->outqueue.right = bridge_buf_add(
    this_bridge->outqueue.right, source->source.x, source->source.y,
    source->dest.x, source->dest.y, NOT_ENCRYPTED);
  if ((from != DOWN) && (find_cable(net, this_bridge->down_r->xloc,
  this_bridge->down_r->yloc, this_bridge->down_r->connections) != NULL))
  {
    this_bridge->outqueue.down = bridge_buf_add(
    this_bridge->outqueue.down, source->source.x, source->source.y,
    source->dest.x, source->dest.y, NOT_ENCRYPTED);
  }
  if ((from != LEFT) && (find_cable(net, this_bridge->left_r->xloc,
  this_bridge->left_r->yloc, this_bridge->left_r->connections) != NULL))
    this_bridge->outqueue.left = bridge_buf_add(
    this_bridge->outqueue.left, source->source.x, source->source.y,
    source->dest.x, source->dest.y, NOT_ENCRYPTED);
}
else
if ((table_find(this_bridge->bridge_table,
                source->dest.y*NETWIDTH+source->dest.x) == UP) &&
    (find_cable(net, this_bridge->up_r->xloc, this_bridge->up_r->yloc,
    this_bridge->up_r->connections) != NULL) &&
    (from != UP))
{ /* Pass to top side of bridge */
  this_bridge->outqueue.up = bridge_buf_add(this_bridge->outqueue.up,
  source->source.x, source->source.y, source->dest.x, source->dest.y,
  NOT_ENCRYPTED);
}
else
if ((table_find(this_bridge->bridge_table,
source->dest.y*NETWIDTH+source->dest.x) == RIGHT) &&
    (find_cable(net, this_bridge->right_r->xloc,
    this_bridge->right_r->yloc, this_bridge->right_r->connections) != NULL) &
    (from != RIGHT))
{ /* Pass to right hand side of bridge */
  this_bridge->outqueue.right = bridge_buf_add(
  this_bridge->outqueue.right, source->source.x, source->source.y,
  source->dest.x, source->dest.y, NOT_ENCRYPTED);
}
else
if ((table_find(this_bridge->bridge_table,
source->dest.y*NETWIDTH+source->dest.x) == DOWN) &&
    (find_cable(net, this_bridge->down_r->xloc,
    this_bridge->down_r->yloc, this_bridge->down_r->connections) !=
    NULL) &&
    (from != DOWN))
{ /* Pass to down side of bridge */
  this_bridge->outqueue.down = bridge_buf_add(this_bridge->outqueue.down,
```

```
    source->source.x, source->source.y, source->dest.x, source->dest.y,
    NOT_ENCRYPTED);
  }
  else
  if ((table_find(this_bridge->bridge_table,
  source->dest.y*NETWIDTH+source->dest.x) == LEFT) &&
      (find_cable(net, this_bridge->left_r->xloc,
      this_bridge->left_r->yloc, this_bridge->left_r->connections) !=
      NULL) &&
      (from != LEFT))
  { /* Pass to left side of bridge */
    this_bridge->outqueue.left = bridge_buf_add(
    this_bridge->outqueue.left, source->source.x, source->source.y,
    source->dest.x, source->dest.y, NOT_ENCRYPTED);
  }
  /* Else message was on correct side (probably) */

  source->last = -1;
}


int maintain_ethernet(netelement net[NETHEIGHT][NETWIDTH],
                      struct icon *icons, int xloc, int yloc)
/* Maintains ethernet objects.  Works out what this device is */
/* and then goes about maintaining it. */
{
  if ((xloc < 0) ||
      (xloc >= NETWIDTH) ||
      (yloc < 0) ||
      (yloc >= NETHEIGHT))
  {
    printf("Not a valid unit to maintain!\n");
    exit(1);
  }

  if (net[yloc][xloc].equipment == EQUIP_SITE)
  { /* Maintain a single site at this spot */
    int old_state_temp = net[yloc][xloc].temp.site->state;
    int old_state_current = net[yloc][xloc].current.site->state;
    int new_state_temp, new_state_current;

    maintain_site(net, net[yloc][xloc].current.site, xloc, yloc, RECEIVER);
    maintain_site(net, net[yloc][xloc].temp.site, xloc, yloc, TRANSMITTER);

    new_state_temp = net[yloc][xloc].temp.site->state;
    new_state_current = net[yloc][xloc].current.site->state;

    /* Redraw the site if the state has changed */
```

```
  if ((new_state_current != old_state_current) ||
      (new_state_temp != old_state_temp))
  {
    struct icon *thisicon = find_net_icon(icons, xloc, yloc);
    if ((new_state_temp == STATE_SEND) ||
        (new_state_temp == STATE_BACKOFF) ||
        (new_state_temp == STATE_SENSE) ||
        (new_state_temp == STATE_JAM))
      draw_site(thisicon->left, thisicon->top,
                thisicon->right-thisicon->left+1,
                thisicon->bottom-thisicon->top+1,
                xloc+yloc*NETWIDTH, new_state_temp);
    else
    if (new_state_current == STATE_RECEIVE)
      draw_site(thisicon->left, thisicon->top,
                thisicon->right-thisicon->left+1,
                thisicon->bottom-thisicon->top+1,
                xloc+yloc*NETWIDTH, new_state_current);
    else /* Display what the transmitter is doing */
      draw_site(thisicon->left, thisicon->top,
                thisicon->right-thisicon->left+1,
                thisicon->bottom-thisicon->top+1,
                xloc+yloc*NETWIDTH, new_state_temp);
  }
}
else
if (net[yloc][xloc].equipment == EQUIP_BRIDGE)
{
  net_bridge *this_bridge = net[yloc][xloc].current.bridge;

  /* Let the four sites of the bridge do their bit */

  maintain_site(net, this_bridge->left_r, xloc, yloc, RECEIVER);
  maintain_site(net, this_bridge->right_r, xloc, yloc, RECEIVER);
  maintain_site(net, this_bridge->up_r, xloc, yloc, RECEIVER);
  maintain_site(net, this_bridge->down_r, xloc, yloc, RECEIVER);

  maintain_site(net, this_bridge->left_t, xloc, yloc, TRANSMITTER);
  maintain_site(net, this_bridge->right_t, xloc, yloc, TRANSMITTER);
  maintain_site(net, this_bridge->up_t, xloc, yloc, TRANSMITTER);
  maintain_site(net, this_bridge->down_t, xloc, yloc, TRANSMITTER);

  /* For each detected transmission:
     1) Update bridge tables.
     2) Possibly pass on message */

  if (this_bridge->left_r->last != -1) /* Incomming msg detected */
    bridge_action(net, this_bridge, LEFT);
```

```
if (this_bridge->right_r->last != -1) /* Incomming msg detected */
  bridge_action(net, this_bridge, RIGHT);

if (this_bridge->up_r->last != -1) /* Incomming msg detected */
  bridge_action(net, this_bridge, UP);

if (this_bridge->down_r->last != -1) /* Incomming msg detected */
  bridge_action(net, this_bridge, DOWN);

/* If there are frames queued up for the sites to */
/* send, then send them, and remove the frames from */
/* the queues. */

if ((this_bridge->up_t->state == STATE_WAIT) &&
    (this_bridge->outqueue.up != NULL))
{
  (void)site_transmit(net, this_bridge->up_t,
                      this_bridge->outqueue.up->source.x,
                      this_bridge->outqueue.up->source.y,
                      this_bridge->outqueue.up->dest.x,
                      this_bridge->outqueue.up->dest.y);
  this_bridge->outqueue.up = bridge_buf_delete(this_bridge->outqueue.up);
}

if ((this_bridge->down_t->state == STATE_WAIT) &&
    (this_bridge->outqueue.down != NULL))
{
  (void)site_transmit(net, this_bridge->down_t,
                      this_bridge->outqueue.down->source.x,
                      this_bridge->outqueue.down->source.y,
                      this_bridge->outqueue.down->dest.x,
                      this_bridge->outqueue.down->dest.y);
  this_bridge->outqueue.down = bridge_buf_delete(this_bridge->outqueue.do
}

if ((this_bridge->left_t->state == STATE_WAIT) &&
    (this_bridge->outqueue.left != NULL))
{
  (void)site_transmit(net, this_bridge->left_t,
                      this_bridge->outqueue.left->source.x,
                      this_bridge->outqueue.left->source.y,
                      this_bridge->outqueue.left->dest.x,
                      this_bridge->outqueue.left->dest.y);
  this_bridge->outqueue.left = bridge_buf_delete(this_bridge->outqueue.le
}

if ((this_bridge->right_t->state == STATE_WAIT) &&
```

```
        (this_bridge->outqueue.right != NULL))
    {
      (void)site_transmit(net, this_bridge->right_t,
                          this_bridge->outqueue.right->source.x,
                          this_bridge->outqueue.right->source.y,
                          this_bridge->outqueue.right->dest.x,
                          this_bridge->outqueue.right->dest.y);
      this_bridge->outqueue.right = bridge_buf_delete(this_bridge->outqueue.rig
    }
  }

  /* Only maintain single sites, and bridges */

  return 0;
}


void update_net(netelement net[NETHEIGHT][NETWIDTH], struct icon *icons)
/* Routine to propagate all the packets, and generally do */
/* everything that needs to be done :-) */
{
  int i, j;

  if (check_net(net) != OK)
  {
    printf("Network error!\n");
    exit(1);
  }

  for (j = 0; j < NETHEIGHT; j++)
    for (i = 0; i < NETWIDTH; i++)
      if (net[j][i].equipment == EQUIP_CABLE)
      {
          /* Delete temp list */
          net[j][i].temp.cable->leftdir =
            list_kill(net[j][i].temp.cable->leftdir, 1);
          net[j][i].temp.cable->rightdir =
            list_kill(net[j][i].temp.cable->rightdir, 1);
          net[j][i].temp.cable->updir =
            list_kill(net[j][i].temp.cable->updir, 1);
          net[j][i].temp.cable->downdir =
            list_kill(net[j][i].temp.cable->downdir, 1);
      }

  /* Move signals along wires */

  for (j = 0; j < NETHEIGHT; j++)
    for (i = 0; i < NETWIDTH; i++)
```

```
  {
    if (net[j][i].equipment == EQUIP_CABLE)
    { /* There is cable here */
      propagate(net, j, i, net[j][i].current.cable->leftdir, LEFT);
      propagate(net, j, i, net[j][i].current.cable->rightdir, RIGHT);
      propagate(net, j, i, net[j][i].current.cable->updir, UP);
      propagate(net, j, i, net[j][i].current.cable->downdir, DOWN);
    }
  }

/* See if sites want to produce packets */

for (j = 0; j < NETHEIGHT; j++)
  for (i = 0; i < NETWIDTH; i++)
    if ((net[j][i].equipment == EQUIP_SITE) ||
        (net[j][i].equipment == EQUIP_BRIDGE))
    { /* See if ethernet device wants to produce a packet.
         If it does, do it */
      maintain_ethernet(net, icons, i, j);
    }

/* Now we need to get rid of all the current packets, and
   replace them with the temp packets. */

for (j = 0; j < NETHEIGHT; j++)
  for (i = 0; i < NETWIDTH; i++)
  {
    if (net[j][i].equipment == EQUIP_CABLE)
    {
      listptr t;

      /* Move temp data to current data */
      t = net[j][i].current.cable->leftdir;
      net[j][i].current.cable->leftdir = net[j][i].temp.cable->leftdir;
      net[j][i].temp.cable->leftdir = t;
      t = net[j][i].current.cable->rightdir;
      net[j][i].current.cable->rightdir = net[j][i].temp.cable->rightdir;
      net[j][i].temp.cable->rightdir = t;
      t = net[j][i].current.cable->updir;
      net[j][i].current.cable->updir = net[j][i].temp.cable->updir;
      net[j][i].temp.cable->updir = t;
      t = net[j][i].current.cable->downdir;
      net[j][i].current.cable->downdir = net[j][i].temp.cable->downdir;
      net[j][i].temp.cable->downdir = t;
    }
  }

/* Display changes */
```

```
for (j = 0; j < NETHEIGHT; j++)
  for (i = 0; i < NETWIDTH; i++)
  {
    struct icon *current = find_net_icon(icons, i, j);

    if (current == NULL)
    {
      printf("Missing icon!\n");
      exit(1);
    }

    if ((net[j][i].equipment == EQUIP_CABLE) &&
        ((sense_cable(net[j][i].current.cable) !=
            sense_cable(net[j][i].temp.cable)) ||
         (fill_style(net[j][i].current.cable) !=
             fill_style(net[j][i].temp.cable))))
    { /* Type of packets on a piece of cable, has changed */
      int id;

      if (net[j][i].equipment == EQUIP_NONE)
        id = NOTHING;
      else
      if (net[j][i].equipment == EQUIP_SITE)
        id = SITE;
      else
      if (net[j][i].equipment == EQUIP_BRIDGE)
        id = BRIDGE;
      else
      if (net[j][i].equipment == EQUIP_CABLE)
        id = net[j][i].current.cable->type;
      else
        printf("Cannot id element!\n");

      if (sense_cable(net[j][i].current.cable) == SENSE_JAM)
        draw_cable(current->left, current->top,
                   current->right-current->left+1,
                   current->bottom-current->top+1, id,
                   COLOUR_PACKETJAM, SOLID_FILL);
      else
      if (sense_cable(net[j][i].current.cable) == SENSE_COLLISION)
        draw_cable(current->left, current->top,
                   current->right-current->left+1,
                   current->bottom-current->top+1, id,
                   COLOUR_PACKETCOLL, SOLID_FILL);
      else
      if (sense_cable(net[j][i].current.cable) == SENSE_DATA)
        draw_cable(current->left, current->top,
```

```
                    current->right-current->left+1,
                    current->bottom-current->top+1, id,
                    COLOUR_PACKET, fill_style(net[j][i].current.cable));
        else
        if (sense_cable(net[j][i].current.cable) == SENSE_NOTHING)
          draw_cable(current->left, current->top,
                    current->right-current->left+1,
                    current->bottom-current->top+1, id,
                    COLOUR_NOTHING, SOLID_FILL);
      }
    }
}


int site_transmit(netelement net[NETHEIGHT][NETWIDTH],
                  net_site *this_site, int sourcex, int sourcey,
                  int destx, int desty)
/* Sets up transmission from one site to another */
{
  if (check_transmission(net, this_site->xloc, this_site->yloc,
                         destx, desty) != OK)
    return ERROR;

  this_site->retry = 0;
  this_site->state = STATE_SENSE;
  this_site->countdown = PACKET_LENGTH;
  this_site->dest.x = destx;
  this_site->dest.y = desty;
  this_site->source.x = sourcex;
  this_site->source.y = sourcey;

  return OK;
}


void fill_mem(int space)
{
  char *prev = (char *)malloc(space);

  if (prev == NULL)
    return;

  while (malloc(space) != NULL); /* Keep mallocing */

  free(prev);
}
```

```
char *describe_data(data_packet *data, char *junk)
{
  sprintf(junk, "%6d %4d ", data->source.y*NETWIDTH+data->source.x,
          data->dest.y*NETWIDTH+data->dest.x);
  if (data->encrypted == ENCRYPTED)
    strcat(junk, "Yes    ");
  else
    strcat(junk, "No     ");

  switch (data->type)
  {
    case HEAD:
      strcat(junk, "Head");
      break;
    case MIDDLE:
      strcat(junk, "Mid");
      break;
    case TAIL:
      strcat(junk, "Tail");
      break;
    case JAM:
      strcat(junk, "JAM");
      break;
  }

  return junk;
}


char *describe_mouse_actions(netelement net[NETHEIGHT][NETWIDTH],
                             struct icon *over, struct icon *currentbutton)
{
  char *result; /* Description of possible action */

  if ((result = (char *)malloc(80*sizeof(char))) == NULL)
    return NULL;

  sprintf(result, "");

  if (over->type == BUTTON)
    if (over->state == PRESSED)
      sprintf(result, "LMB: deselect toggle switch");
    else
      switch (over->id)
      {
        case HORTTAP:
        case HORBTAP:
        case HOR:
```

```
      case VERLTAP:
      case VERRTAP:
      case VER:
      case LT:
      case LB:
      case RT:
      case RB:
        sprintf(result, "LMB: select cable");
        break;
      case SITE:
        sprintf(result, "LMB: select site");
        break;
      case BRIDGE:
        sprintf(result, "LMB: select bridge");
        break;
      case NOTHING:
        sprintf(result, "LMB: select bulldozer");
        break;
      default:
        sprintf(result, "<unknown!>");
    }
else
if (over->type == MOMENT)
  switch (over->id)
  {
    case TIME_FASTER:
      sprintf(result, "LMB: increase time rate");
      break;
    case TIME_SLOWER:
      sprintf(result, "LMB: decrease time rate");
      break;
    case TIME_STEP:
      sprintf(result, "LMB: step one time unit");
      break;
    case TIME_STOP:
      sprintf(result, "LMB: halt time");
      break;
    case QUIT:
      sprintf(result, "LMB: quit SimEther");
      break;
    case LOAD:
      sprintf(result, "LMB: load a new network");
      break;
    case SAVE:
      sprintf(result, "LMB: save this network");
      break;
    default:
      sprintf(result, "!!!");
```

```
    }
  else
  if (over->type == NETELEMENT)
  {
    if (net[over->aux][over->state].equipment == EQUIP_NONE)
      if ((currentbutton == NULL) || (currentbutton->id == NOTHING));
      /* nothing */
      else
        sprintf(result, "LMB: build");
    else
    if (net[over->aux][over->state].equipment == EQUIP_SITE)
      if ((currentbutton != NULL) && (currentbutton->id == NOTHING))
        sprintf(result, "LMB: bulldoze site");
      else
        sprintf(result, "LMB: snoop on site  RMB: data source/dest");
    else
    if (net[over->aux][over->state].equipment == EQUIP_BRIDGE)
      if ((currentbutton != NULL) && (currentbutton->id == NOTHING))
        sprintf(result, "LMB: bulldoze bridge");
      else
        sprintf(result, "LMB: examine bridge tables");
    else
    if (net[over->aux][over->state].equipment == EQUIP_CABLE)
      if ((currentbutton != NULL) && (currentbutton->id == NOTHING))
        sprintf(result, "LMB: bulldoze cable");
      else
        sprintf(result, "LMB: snoop at data");
    else
      sprintf(result, "something else!");
  } /* End of over a network element */

  return result;
}
```

D.15 clock.hpp

```
class clock
{
  int xloc, yloc, xsize, ysize;
  int minutes, seconds;
  int oldminutes, oldseconds;
```

```
  int border; /* border space between face and edge of icon */

  public:
  clock();
  ~clock();
  void init(int, int, int, int);
  void update();
  void display();

  private:
  void draw_hands(int, int, int);
};
```

D.16 clock.cpp

```
#include "clock.hpp"

clock::clock()
{
  minutes = 0;
  seconds = 0;
  oldminutes = 0;
  oldseconds = 0;
}

clock::~clock()
{
}

void clock::init(int x, int y, int xs, int ys)
{
  xloc = x;
  yloc = y;
  xsize = xs;
  ysize = ys;

  border = xsize/15;

  /* Clear screen area, and draw face of clock */
  setfillstyle(SOLID_FILL, COLOUR_BUTTON);
  bar(xloc, yloc, xloc+xsize-1, yloc+ysize-1);
  setcolor(COLOUR_SHADOW);
  line(xloc, yloc+ysize-1, xloc+xsize-1, yloc+ysize-1);
```

```
  line(xloc+xsize-1, yloc+ysize-1, xloc+xsize-1, yloc);
  setcolor(COLOUR_SUN);
  line(xloc, yloc+ysize-1, xloc, yloc);
  line(xloc, yloc, xloc+xsize-1, yloc);

  setcolor(COLOUR_TEXT);
  ellipse(xloc+xsize/2, yloc+ysize/2, 0, 360,
          xsize/2-border, ysize/2-border);
}

void clock::update()
{
  oldminutes = minutes;
  oldseconds = seconds;

  seconds = seconds + 5;

  if (seconds > 59)
  {
    seconds = 0;
    minutes++;
    if (minutes > 59)
    minutes = 0;
  }
}

void clock::display()
{
  /* Erase old hand positions */
  draw_hands(oldminutes, oldseconds, COLOUR_BUTTON);
  /* Draw new hand positions */
  draw_hands(minutes, seconds, COLOUR_TEXT);
}

void clock::draw_hands(int min, int sec, int colour)
{
  int x, y;
  double angle; /* Angle clockwise from 12 o'clock, in radians */

  setcolor(colour);

  angle = (double)min*6.0/180.0*3.1415;

  /* Draw the minute hand */
  x = sin(angle)*(xsize/2-3*border)+xloc+xsize/2;
  y = -cos(angle)*(ysize/2-3*border)+yloc+ysize/2;
  line(x, y, xloc+xsize/2, yloc+ysize/2);
```

```
  angle = (double)sec*6.0/180.0*3.1415;

  /* draw the second hand */
  x = sin(angle)*(xsize/2-2*border)+xloc+xsize/2;
  y = -cos(angle)*(ysize/2-2*border)+yloc+ysize/2;
  line(x, y, xloc+xsize/2, yloc+ysize/2);
}
```

D.17 request.hpp

```
class requestor
{
  int xs, ys;
  struct mem_list *memory, *tail;
  int size;
  int l, t;
  int height; /* Amount of shadow to produce under window */
  Mstatus position;
  int leftl, leftr, leftt, leftb;
  int i, j;
  int gran_size; /* Dimension size of screen for memory allocation */

  public:
  requestor();
  ~requestor();
  int init(int, int, int, int);
  int bordersize();
  void close();

  private:
};
```

D.18 request.cpp

```
struct mem_list
{
  void far *mem;
  struct mem_list *next;
```

```
};

#include "request.hpp"

requestor::requestor()
{
  memory = NULL;
  tail = NULL;
  gran_size = 20; /* Dimension in pixels */
}

void requestor::~requestor()
{
}

int requestor::init(int low, int tow, int hor_size, int ver_size)
{
  /* May scale window to fit on screen */

  l = low; /* Left of window */
  t = tow; /* Top of window */
  xs = hor_size;
  ys = ver_size;

  if (l < 0)
    l = 0;
  if (t < 0)
    t = 0;
  if (l+xs-1 > getmaxx())
    xs = getmaxx()-l+1;
  if (t+ys-1 > getmaxy())
    ys = getmaxy()-t+1;

  height = ys/15;

  gmouse.Mshow(0);

  /* Work out the memory requirements */

  for (j = t; j <= t+ys-1; j = j+gran_size)
    for (i = l; i <= l+xs-1; i = i+gran_size)
    {
      struct mem_list *new_entry;
      void far *temp;
      int r = i+gran_size-1, b = j+gran_size-1;
      if (r > getmaxx())
        r = getmaxx();
      if (b > getmaxy())
```

```
        b = getmaxy();

      size = imagesize(i, j, r, b);
      if ((temp = farmalloc(size)) == NULL)
      {
        close();
        return ERROR;
      }
      getimage(i, j, r, b, temp);

      if ((new_entry = (struct mem_list *)malloc(sizeof(struct mem_list)))
          == NULL)
        return NULL;

      new_entry->mem = temp;
      new_entry->next = NULL;

      if (memory != NULL)
      {
        tail->next = new_entry;
        tail = new_entry;
      }
      else
      {
        memory = new_entry;
        tail = new_entry;
      }
    }

  /* Draw requestor */
  setfillstyle(SOLID_FILL, COLOUR_DESKTOP); /* Draw requestor surface */
  bar(l+1, t+1, l+xs-height-2, t+ys-height-2);

  moveto(l, t+ys-1-height);
  setcolor(COLOUR_SHADOW); /* Bottom and right */
  lineto(l+xs-1-height, t+ys-1-height);
  lineto(l+xs-1-height, t);
  setcolor(COLOUR_SUN); /* Top and left */
  lineto(l, t);
  lineto(l, t+ys-1-height);
  setfillstyle(SOLID_FILL, COLOUR_TEXT); /* base and side shadow */
  bar(l+height, t+ys-height, l+xs-1, t+ys-1);
  bar(l+xs-height, t+height, l+xs-1, t+ys-1);

  return OK;
}

int requestor::bordersize()
```

```
{
  return height;
}

void requestor::close()
{
  struct mem_list *temp;

  /* Restore the screen */

  for (j = t; j <= t+ys-1; j = j+gran_size)
    for (i = l; i <= l+xs-1; i = i+gran_size)
      if ((memory != NULL) && (memory->mem != NULL))
      {
        putimage(i, j, memory->mem, COPY_PUT);
        temp = memory;
        memory = memory->next;
        farfree(temp->mem);
        free(temp);
      }
}
```

D.19 window.hpp

```
class textwindow
{
  int xloc, yloc, xdim, ydim;
  int lineshigh; /* Number of lines of text in this window */
  int fontsize;
  int fontgap; /* Space between lines of text */
  int bordersize;
  int outerright, outerbottom, innerleft, innerright, innertop, innerbottom;
  int currentline; /* The line the cursor is on */
  char *title;
  int maxlines; /* the maximum number of lines we'll ever need */
  char **textlist; /* Array of pointers to text - one per line */

  public:
    textwindow();
    ~textwindow();
    void init(int, int, int, int, char *);
    void clear();
```

```
    void display(char *);
};
```

D.20 window.cpp

```
#include "window.hpp"

void textwindow::textwindow()
{
  int i;

  maxlines = 132; /* Allow up to this many lines of text */

  if ((textlist = (char **)malloc(maxlines*sizeof(char *))) == NULL)
    return;

  for (i = 0; i < maxlines; i++)
    textlist[i] = NULL;
}

void textwindow::~textwindow()
{
  int i;

  if (textlist != NULL)
  {
    for (i = 0; i < maxlines; i++)
      if (textlist[i] != NULL)
        free(textlist[i]);

    free(textlist);
  }
}

void textwindow::init(int outerleft, int outertop, int xsize, int ysize,
                      char *text)
{
  bordersize = getmaxx()/100;
  /* How much to bring things in from the outside */

  settextstyle(DEFAULT_FONT, HORIZ_DIR, 0);
  fontsize = textheight("A"); /* Height of arbitrary tall character */
  fontgap = fontsize/4;
```

```
outerright = outerleft+xsize-1;
if (outerright > getmaxx())
  outerright = getmaxx();

outerbottom = outertop+ysize-1;
if (outerbottom > getmaxy())
  outerbottom = getmaxy();

innerleft = outerleft+bordersize-1;
innerright = outerright-bordersize+1;
innertop = outertop+fontsize+2;
innerbottom = outerbottom-bordersize+1;
lineshigh = (innerbottom-innertop-1-2*bordersize)/(fontsize+fontgap);
/* Lines of text that will fit */

xloc = outerleft; /* Initialise object variables */
yloc = outertop;
xdim = xsize;
ydim = ysize;
title = text;
currentline = 1; /* The line (1 upwards) we are going to print on */

if (textlist == NULL) /* There was not enough mem for text */
  return;

/* draw the window */
setcolor(COLOUR_SHADOW);
line(outerleft, outerbottom, outerright, outerbottom);
line(outerright, outerbottom, outerright, outertop);
setcolor(COLOUR_SUN);
line(outerleft, outerbottom, outerleft, outertop);
line(outerleft, outertop, outerright, outertop);
line(innerleft, innerbottom, innerright, innerbottom);
line(innerright, innerbottom, innerright, innertop);
setcolor(COLOUR_SHADOW);
line(innerleft, innerbottom, innerleft, innertop);
line(innerleft, innertop, innerright, innertop);
setfillstyle(SOLID_FILL, COLOUR_DESKTOP);
bar(outerleft+1, outertop+1, innerright, innertop-1);
bar(innerright+1, outertop+1, outerright-1, innerbottom);
bar(innerleft, innerbottom+1, outerright-1, outerbottom-1);
bar(outerleft+1, innertop, innerleft-1, outerbottom-1);
setfillstyle(SOLID_FILL, COLOUR_TEXTWINDOW);
bar(innerleft+1, innertop+1, innerright-1, innerbottom-1);
setcolor(COLOUR_TEXT);
settextjustify(LEFT_TEXT, TOP_TEXT);
outtextxy(innerleft+1, outertop+2, text);
```

```
}

void textwindow::clear()
{
}

void textwindow::display(char *newline)
{
  int i; /* Dummy loop variable */
  struct viewporttype oldvp;

  if ((textlist == NULL) || (newline == NULL))
    return;

  getviewsettings(&oldvp); /* Record the old viewport */

  settextstyle(DEFAULT_FONT, HORIZ_DIR, 0);
  settextjustify(LEFT_TEXT, TOP_TEXT);
  setviewport(innerleft+bordersize, innertop+bordersize+1, innerright-1,
              innerbottom-1, 1);

  if (currentline > lineshigh) /* We need to scroll first */
  {
    for (i = 0; i <= lineshigh-1; i++)
    {
      setcolor(COLOUR_TEXTWINDOW);
      outtextxy(0, (fontsize+fontgap)*i, textlist[i]);
      if (i == 0)
        free(textlist[0]);
      if (i < lineshigh-1)
      {
        textlist[i] = textlist[i+1];
        setcolor(COLOUR_TEXT);
        outtextxy(0, (fontsize+fontgap)*i, textlist[i]);
      }
    }
    currentline = lineshigh;
  }

  if ((textlist[currentline-1] = (char *)malloc(strlen(newline)+1)) == NULL)
  { /* Try and exit gracefully on error */
    setviewport((&oldvp)->left, (&oldvp)->top, (&oldvp)->right,
                (&oldvp)->bottom, (&oldvp)->clip);
                /* Revert back to old viewport */
    return; /* Error */
  }

  strcpy(textlist[currentline-1], newline);
```

```
  currentline++;

  setcolor(COLOUR_TEXT);
  outtextxy(0, (fontsize+fontgap)*(currentline-2), textlist[currentline-2]);

  setviewport((&oldvp)->left, (&oldvp)->top, (&oldvp)->right,
              (&oldvp)->bottom, (&oldvp)->clip);
              /* Revert back to old viewport */
}
```

D.21 mouse.i

```
/* MOUSE.I: Mouse interface object. */

#include <dos.h>
#include <stddef.h>

#define call_mouse int86(0x33, &inreg, &outreg)
#define EVENTMASK 0x54
#define lower (x, y) (x < y)? x : y
#define upper (x, y) (x > y)? x : y
#define ButtonL   0
#define ButtonR   1
#define ButtonM   2
#define SOFTWARE  0
#define HARDWARE  1
#define FALSE   0
#define TRUE    1
#define OFF   0
#define ON    1

union REGS inreg, outreg;

typedef struct
{
  int present, buttons;
} Mresult;

typedef struct
{
  int button_status, button_count, xaxis, yaxis;
} Mstatus;
```

```
typedef struct
{
  int x_count, y_count;
} Mmovement;

typedef struct
{
  unsigned flag, button, xaxis, yaxis;
} mouse_event;

typedef struct
{
  unsigned int ScreenMask[16], CursorMask[16], xkey, ykey;
} g_cursor;

class Mouse
{
  int Mview;

  protected:
    Mouse();
    ~Mouse();
  public:
    static mouse_event far *Mevents;
    Mmovement *Mmotion();
    Mresult *Mreset();
    Mstatus Mpos();
    Mstatus Mpressed(int button);
    Mstatus Mreleased(int button);
    void Mshow(int showstat);
    void Mmoveto(int xaxis, int yaxis);
    void Mxlimit(int min_x, int max_x);
    void Mylimit(int min_y, int max_y);
    void Mmove_ratio(int xsize, int ysize);
    void Mspeed(int speed);
    void Mconceal(int left, int top, int right, int bottom);
};

class GMouse : public Mouse
{
  private:
    void set_cursor(int xaxis, int yaxis, unsigned mask_Seg,
                    unsigned mask_Ofs);
  public:
    void Set_Cursor(g_cursor ThisCursor);
    void Mlightpen(int set);
};
```

```
class TMouse : public Mouse
{
  public:
    void Set_Cursor(int cursor_type, unsigned s_start, unsigned s_stop);
    void Mlightpen(int set);
};


/* Standard mouse functions */

Mouse::Mouse()
{
}

Mouse::~Mouse()
{
}

Mresult *Mouse::Mreset()
{
  static Mresult m;

  Mview = OFF;
  inreg.x.ax = 0;
  call_mouse;
  m.present = outreg.x.ax;
  m.buttons = outreg.x.bx;
  if (m.present)
    Mshow (TRUE);
  return (&m);
}

void Mouse::Mshow(int showstat)
{
  if (showstat)
  {
    inreg.x.ax = 1;
    if (!Mview)
      call_mouse;
    Mview = ON;
  }
  else
  {
    inreg.x.ax = 2;
    if (Mview)
      call_mouse;
    Mview = OFF;
  }
```

```
}

Mstatus Mouse::Mpos()
{
  static Mstatus m;

  inreg.x.ax = 3;
  call_mouse;
  m.button_status = outreg.x.bx;
  m.xaxis = outreg.x.cx;
  m.yaxis = outreg.x.dx;
  return (m);
}

void Mouse::Mmoveto(int xaxis, int yaxis)
{
  inreg.x.ax = 4;
  inreg.x.cx = xaxis;
  inreg.x.dx = yaxis;
  call_mouse;
}

Mstatus Mouse::Mpressed(int button)
{
  static Mstatus m;

  inreg.x.ax = 5;
  inreg.x.bx = button;
  call_mouse;
  m.button_status = outreg.x.ax;
  m.button_count = outreg.x.bx;
  m.xaxis = outreg.x.cx;
  m.yaxis = outreg.x.dx;
  return (m);
}

Mstatus Mouse::Mreleased(int button)
{
  static Mstatus m;

  inreg.x.ax = 6;
  inreg.x.bx = button;
  call_mouse;
  m.button_status = outreg.x.ax;
  m.button_count = outreg.x.bx;
  m.xaxis = outreg.x.cx;
  m.yaxis = outreg.x.dx;
  return (m);
```

```
}

void Mouse::Mxlimit(int min_x, int max_x)
{
  inreg.x.ax = 7;
  inreg.x.cx = min_x;
  inreg.x.dx = max_x;
  call_mouse;
}

void Mouse::Mylimit(int min_y, int max_y)
{
  inreg.x.ax = 8;
  inreg.x.cx = min_y;
  inreg.x.dx = max_y;
  call_mouse;
}

void GMouse::set_cursor(int xaxis, int yaxis, unsigned mask_Seg,
                        unsigned mask_Ofs)
{
  struct SREGS seg;

  inreg.x.ax = 9;
  inreg.x.bx = xaxis;
  inreg.x.cx = yaxis;
  inreg.x.dx = mask_Ofs;
  seg.es     = mask_Seg;
  int86x(0x33, &inreg, &outreg, &seg);
}

void TMouse::Set_Cursor(int cursor_type, unsigned s_start, unsigned s_stop)
{
  inreg.x.ax = 10;
  inreg.x.bx = cursor_type;
  inreg.x.cx = s_start;
  inreg.x.dx = s_stop;
  call_mouse;
}

Mmovement* Mouse::Mmotion()
{
  static Mmovement m;

  inreg.x.ax = 11;
  call_mouse;
  m.x_count = _CX;
  m.y_count = _DX;
```

```
  return (&m);
}

void GMouse::Mlightpen(int set)
{
  if (set)
    inreg.x.ax = 13;
  else
    inreg.x.ax = 14;
  call_mouse;
}

void TMouse::Mlightpen(int set)
{
  if (set)
    inreg.x.ax = 13;
  else
    inreg.x.ax = 14;
  call_mouse;
}

void Mouse::Mmove_ratio(int xsize, int ysize)
{
  inreg.x.ax = 15;
  inreg.x.cx = xsize;
  inreg.x.dx = ysize;
  call_mouse;
}

void Mouse::Mconceal(int left, int top, int right, int bottom)
{
  inreg.x.ax = 16;
  inreg.x.cx = left;
  inreg.x.dx = top;
  inreg.x.si = right;
  inreg.x.di = bottom;
  call_mouse;
}

void Mouse::Mspeed(int speed)
{
  inreg.x.ax = 19;
  inreg.x.dx = speed;
  call_mouse;
}

void GMouse::Set_Cursor(g_cursor ThisCursor)
{
```

```
  set_cursor(ThisCursor.xkey, ThisCursor.ykey, _DS,
             (unsigned)ThisCursor.ScreenMask);
}


GMouse gmouse;
TMouse tmouse;
```

Appendix E

Contacting the author

If you are interested in SimEther, feel free to contact me at one of the following addresses:

Internet: vanz@tragula.equinox.gen.nz or
Martin_Nieuwelaar@equinox.gen.nz

Postal: 15 Doncaster St
Christchurch
New Zealand 8004

My supervisor, Ray Hunt, can be contacted at the following addresses:

Internet: ray@cosc.canterbury.ac.nz

Dialcom: 6401:TZQ191

Fax: +64-3-364-2999
Work ph: +64-3-364-2347

Appendix F

Trademarks

Amiga is a trademark of Commodore-Amiga, Inc.

Macintosh is a trademark licenced to Apple Computer, Inc.

Borland C++ V3.0 is copyright 1990, 1991 by Borland International Inc.

IBM is a registered trademark of International Business Machines Corporation.

Microsoft, MS, and MS-DOS are registered trademarks, and Windows is a trademark of Microsoft Corporation.

UNIX is a registered trademark of UNIX Systems Laboratories.